

# **IMPLEMENTASI DETEKSI DAN KOREKSI *ERROR* PADA KOMUNIKASI SERIAL ARDUINO BERBASIS UART DENGAN METODE *HAMMING CODE***

## **SKRIPSI**

Untuk memenuhi sebagian persyaratan  
memperoleh gelar Sarjana Teknik

Disusun oleh:  
Anggi Fajar Andana  
NIM: 145150301111022



PROGRAM STUDI TEKNIK KOMPUTER  
JURUSAN TEKNIK INFORMATIKA  
FAKULTAS ILMU KOMPUTER  
UNIVERSITAS BRAWIJAYA  
MALANG  
2018

# **IMPLEMENTASI DETEKSI DAN KOREKSI *ERROR* PADA KOMUNIKASI SERIAL ARDUINO BERBASIS UART DENGAN METODE *HAMMING CODE***

## **SKRIPSI**

Untuk memenuhi sebagian persyaratan  
memperoleh gelar Sarjana Teknik

Disusun oleh:  
Anggi Fajar Andana  
NIM: 145150301111022



PROGRAM STUDI TEKNIK KOMPUTER  
JURUSAN TEKNIK INFORMATIKA  
FAKULTAS ILMU KOMPUTER  
UNIVERSITAS BRAWIJAYA  
MALANG  
2018

## PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 05 Juli 2018



Anggi Fajar Andana

NIM: 145150301111022

## KATA PENGANTAR

Dengan menyebut nama Allah SWT yang Maha Pengasih dan Maha Penyayang, penulis memanjatkan puji syukur karena rahmatnya yang melimpah sehingga laporan skripsi yang berjudul “IMPLEMENTASI DETEKSI DAN KOREKSI *ERROR* PADA KOMUNIKASI SERIAL ARDUINO BERBASIS UART DENGAN METODE *HAMMING CODE*” ini dapat terselesaikan. Atas banyak dukungan dan bantuan yang diberikan dalam penyusunan skripsi ini, maka penulis ingin menyampaikan rasa hormat dan terima kasih kepada:

1. Ibu Nurwidayati selaku orangtua penulis, yang telah sabar mendidik dan membesarkan penulis, serta memberikan doa dan semangat terus-menerus demi terselesaikannya skripsi ini.
2. Bapak Sabriansyah Rizqika Akbar, S.T, M.Eng. dan Bapak Rizal Maulana , S.T., M.T., M.Sc. selaku dosen pembimbing skripsi penulis yang membimbing dan mengarahkan penulis sehingga dapat menyelesaikan skripsi ini.
3. Bapak Tri Astoto Kurniawan, S.T, M.T, Ph.D selaku Ketua Jurusan Teknik Informatika.
4. Seluruh civitas akademika Fakultas Ilmu Komputer Universitas Brawijaya yang telah banyak memberi bantuan dan dukungan kepada penulis selama menempuh studi Informatika di Universitas Brawijaya dan selama penyelesaian skripsi ini.
5. Sahabat kecil penulis Anida, Findie, Asha, dan Hilma yang sudah banyak memberikan doa dan semangat.
6. Sahabat penulis Reynald, Devi, Ganda, Lutfi, Wisnu, Ikhwan, Khurin, Fanani, Sandi, Mila dan Dgengz, Misbah, Cindy, Okky, Reno, Ayu dan Keluarga Besar Teknik Komputer khususnya angkatan 2014 yang sudah memberikan dukungan.

Penulis menyadari dalam penyusunan skripsi ini masih belum sempurna dan masih memiliki banyak kekurangan, oleh karena itu saran dan kritik sangat dibutuhkan untuk penyempurnaan penelitian ini. Akhir kata penulis berharap semoga skripsi ini dapat memberikan banyak manfaat bagi semua pihak yang menggunakannya.

Malang, 05 Juli 2018

Penulis

anggiifaa@gmail.com



## DAFTAR GAMBAR

Gambar 2.1 Model komunikasi digital .....	7
Gambar 2.2 Model <i>Single-bit Error</i> .....	8
Gambar 2.3 Model <i>Burst Error</i> .....	8
Gambar 2.4 Pengiriman data pada komunikasi UART .....	9
Gambar 2.5 Model komunikasi digital yang terdapat <i>noise</i> .....	10
Gambar 2.6 Mikrokontroler <i>Arduino Uno</i> .....	13
Gambar 2.7 <i>Arduino IDE</i> .....	14
Gambar 2.8 Sensor DHT11 .....	14
Gambar 3.1 Diagram alir metodologi penelitian .....	16
Gambar 3.2 Diagram blok perancangan sistem .....	18
Gambar 3.3 Flowchart proses <i>encode</i> data .....	19
Gambar 3.4 Flowchart proses <i>decode</i> data .....	20
Gambar 3.5 Flowchart Pengujian <i>Error Data</i> .....	21
Gambar 5.1 Diagram Blok perancangan sistem <i>encode</i> dan <i>decode</i> data .....	27
Gambar 5.2 Skematik perancangan sensor DHT11 dan <i>Arduino Uno</i> .....	28
Gambar 5.3 Skematik perancangan antar <i>Arduino Uno</i> .....	29
Gambar 5.4 Skematik perancangan sistem pengujian .....	30
Gambar 5.5 Flowchart <i>encode</i> pengiriman data .....	33
Gambar 5.6 Flowchart penambahan <i>parity bit</i> dengan metode <i>Hamming Code</i> .....	34
Gambar 5.7 Flowchart perancangan metode <i>Hamming Code</i> pada sistem pengujian bagian pengirim .....	37
Gambar 5.8 Flowchart <i>decode</i> pada penerima data .....	39
Gambar 5.9 Flowchart deteksi <i>error</i> pada proses <i>decode</i> data dengan metode <i>Hamming Code</i> .....	40
Gambar 5.10 Flowchart koreksi <i>error</i> .....	41
Gambar 5.11 Flowchart konversi desimal ke biner .....	42
Gambar 5.12 Flowchart sistem pengujian data .....	46
Gambar 5.13 Implementasi sensor DHT11 pada <i>Arduino Uno</i> .....	49
Gambar 5.14 Implementasi antar mikrokontroler <i>Arduino Uno</i> .....	50
Gambar 5.15 Implementasi sistem pengujian .....	50
Gambar 6.1 Pengujian fungsional antar <i>arduino uno</i> .....	67

Gambar 6.2 Pengujian data suhu.....	68
Gambar 6.3 Tampilan pengujian fungsional .....	68
Gambar 6.4 Menu pengujian data <i>error</i> pada serial monitor .....	73
Gambar 6.5 Data pengujian pertama .....	74
Gambar 6.6 Data pengujian kedua .....	78



## DAFTAR ISI

PENGESAHAN .....	ii
PERNYATAAN ORISINALITAS .....	iii
KATA PENGANTAR.....	iv
ABSTRAK.....	v
ABSTRACT.....	vi
DAFTAR ISI.....	vii
DAFTAR TABEL.....	x
DAFTAR GAMBAR.....	xiii
DAFTAR LAMPIRAN .....	xv
BAB 1 PENDAHULUAN.....	1
1.1 Latar belakang.....	1
1.2 Rumusan masalah .....	2
1.3 Tujuan .....	3
1.4 Manfaat.....	3
1.5 Batasan masalah .....	3
1.6 Sistematika pembahasan.....	4
BAB 2 LANDASAN KEPUSTAKAAN .....	6
2.1 Tinjauan Pustaka .....	6
2.2 Dasar Teori .....	7
2.2.1 Komunikasi Digital.....	7
2.2.2 Jenis-jenis <i>Error</i> .....	8
2.2.3 Komunikasi UART .....	9
2.2.4 <i>Error Control Coding</i> .....	10
2.2.5 <i>Hamming Code</i> .....	10
2.2.6 Mikrokontroller <i>Arduino Uno</i> .....	13
2.2.7 Sensor DHT11 .....	14
BAB 3 METODOLOGI .....	16
3.1 Studi Literatur .....	16
3.2 Analisis Kebutuhan .....	17
3.2.1 Kebutuhan Perangkat Keras dan Perangkat Lunak.....	17
3.2.2 Kebutuhan Fungsional .....	17



3.3 Perancangan .....	18
3.4 Implementasi .....	18
3.4.1 Implementasi Pada <i>Arduino Uno</i> (pengirim) .....	18
3.4.2 Implementasi Pada Penerima .....	19
3.5 Pengujian dan Analisis .....	21
3.6 Kesimpulan dan Saran .....	22
<b>BAB 4 REKAYASA KEBUTUHAN .....</b>	<b>23</b>
4.1 Gambaran Umum Sistem .....	23
4.2 Batasan Sistem .....	23
4.3 Analisis Kebutuhan .....	24
4.3.1 Kebutuhan Fungsional .....	24
4.3.2 Kebutuhan Non Fungsional .....	24
4.3.2.1 Kebutuhan Implementasi Metode Hamming Code .....	24
4.3.2.2 Kebutuhan Perangkat Keras .....	25
4.3.3 Kebutuhan Perangkat Lunak .....	26
<b>BAB 5 PERANCANGAN DAN IMPLEMENTASI .....</b>	<b>27</b>
5.1 Perancangan Sistem .....	27
5.1.1 Perancangan Perangkat Keras .....	27
5.1.1.1 Perancangan DHT11 dan <i>Arduino Uno</i> .....	28
5.1.1.2 Perancangan Antar Mikrokontroler <i>Arduino Uno</i> .....	29
5.1.1.3 Perancangan Sistem Pengujian .....	30
5.1.2 Perancangan Perangkat Lunak .....	31
5.1.2.1 Perancangan Antarmuka Serial .....	32
5.1.2.2 Perancangan Metode Hamming Code Pada Pengirim .....	32
5.1.2.3 Perancangan Metode Hamming Code Pada Pengiriman Data .....	32
5.1.2.4 Perancangan Metode Hamming Code Pada Sistem Pengujian Data Error .....	37
5.1.2.5 Perancangan Pada Sistem Pengujian .....	38
5.2 Implementasi .....	48
5.2.1 Implementasi Perangkat Keras .....	48
5.2.1.1 Implementasi Sensor DHT11 Pada <i>Arduino Uno</i> .....	48
5.2.1.2 Implementasi Antar Mikrokontroler <i>Arduino Uno</i> .....	49

5.2.1.3 Implementasi Sistem Pengujian.....	50
5.2.2 Implementasi Perangkat Lunak.....	50
5.2.2.1 Implementasi Antar Muka Serial Pada Pengirim dan Penerima Data .....	50
5.2.2.2 Implementasi Metode Hamming Code pada encode data ....	51
5.2.2.3 Implementasi Metode Hamming Code Pada Decode Data ...	55
5.2.2.4 Implementasi Metode Hamming Code Pada Penerima Data	55
5.2.2.5 Impelementasi Metode Hamming Code Pada Sistem Pengujian .....	60
BAB 6 PENGUJIAN DAN ANALISIS.....	67
6.1 Pengujian Fungsional .....	67
6.1.1 Tujuan Pengujian .....	67
6.1.2 Prosedur Pengujian.....	67
6.1.3 Hasil dan Analisis Pengujian.....	68
6.2 Pengujian Non Fungsional.....	68
6.2.1 Tujuan Pengujian .....	68
6.2.2 Prosedur Pengujian.....	68
6.2.3 Hasil dan Analisis Pengujian.....	69
6.3 Pengujian Data <i>Error</i> .....	73
6.3.1 Tujuan Pengujian .....	73
6.3.2 Prosedur Pengujian.....	73
6.3.3 Hasil dan Analisis Pengujian.....	74
6.4 Pengujian Waktu Metode <i>Hamming Code</i> .....	82
6.4.1 Tujuan Pengujian .....	82
6.4.2 Prosedur Pengujian.....	82
6.4.3 Hasil dan Analisis Pengujian.....	82
6.4.3.1 Pengujian Waktu <i>Encode Data</i> .....	83
6.4.3.2 Pengujian Waktu <i>Decode Data</i> .....	86
BAB 7 PENUTUP .....	90
7.1 Kesimpulan.....	90
7.2 Saran .....	91
DAFTAR PUSTAKA.....	92
LAMPIRAN A KODE PROGRAM.....	93

## DAFTAR LAMPIRAN

A.1 KODE PROGRAM <i>ENCODE</i> DATA PADA BAGIAN PENGIRIM .....	93
A.2 KODE PROGRAM <i>DECODE</i> PADA BAGIAN PENERIMA .....	96
A.3 KODE PROGRAM SISTEM PENGUJIAN PADA BAGIAN PENGIRIM .....	99
A.4 KODE PROGRAM SISTEM PENGUJIAN PADA BAGIAN PENERIMA .....	100





## BAB 1 PENDAHULUAN

### 1.1 Latar belakang

Komunikasi data memiliki peranan penting bagi pertukaran data antar perangkat atau media transmisi. Komunikasi data merupakan suatu cara untuk menyebarkan dan menyampaikan data atau informasi dengan menggunakan media transmisi data. Proses komunikasi akan melakukan pengiriman data atau pesan antar perangkat yang bersangkutan. Perangkat harus saling terhubung antara perangkat keras/*hardware* atau perangkat lunak/*software*.

Kepentingan komunikasi data terletak pada kelancaran penyampaian dan penyebaran datanya. Kelancaran penyampaian dan penyebaran data akan membawa dampak pada kelancaran suatu proses komunikasi yang bergantung pada data dan informasi (Lubis, et al., 2012). Namun, pada saat pengiriman data atau pesan yang ditransmisikan melalui sistem komunikasi dapat terpengaruh oleh kebisingan (*noise*) yang dapat menyebabkan terjadinya *error* (Muhajir, et al., 2016). Penyebab *error* dapat disebabkan oleh banyak hal seperti gangguan cuaca, lingkungan, usia perangkat, dll. Perangkat yang mengalami *error*, dapat salah menyampaikan informasi atau bahkan tidak dapat menyampaikan informasi. Berdasarkan permasalahan tersebut, munculah konsep *Error Detection* dan *Error Correction*. Konsep *Error Detection And Error Correction* banyak diterapkan dalam *encoding* dan *decoding* transmisi data.

Terdapat beberapa metode *Error Detection And Error Correction* yang diterapkan dalam proses *Error Control Coding*. Antara lain dengan menambahkan perangkat tambahan yaitu *Hardware Redudancy*, informasi tambahan yang disebut dengan *Information Redudancy*, dan waktu tambahan yaitu *Time Redudancy*. Pada penelitian ini, penulis menggunakan konsep *Information Redudancy* dengan menambahkan informasi tambahan pada objek penelitian. Konsep *Information Redudancy* memiliki beberapa metode dalam menangani kasus *error* data yang muncul. Metode-metode tersebut antara lain, *Cyclic Redudancy Check* dengan menambahkan *parity bit* menggunakan *generator* sebagai acuan pada bagian pengirim dan penerima. *Linear Feedback Shift Register* (LFSR) menggunakan *shift register* dan logika XOR dalam penerapannya. Serta contoh terakhir yaitu metode *Hamming Code* menggunakan logika XOR dalam membuat *parity bit* tambahan yang ditambahkan. Kelebihan metode ini antara lain sangat efektif jika digunakan untuk melakukan deteksi dan koreksi *single bit error* serta untuk deteksi *burst error* yang akan dijadikan objek penelitian. Kemudian keuntungan lain yang didapatkan yaitu cara kerjanya yang cukup sederhana dan tidak membutuhkan alokasi memori yang banyak (Lubis, et al., 2012). *Hamming Code* mampu mendeteksi beberapa *error*, namun hanya dapat mengoreksi satu *error* (*single Error Correction*). Sehingga metode ini sangat cocok digunakan untuk deteksi *error* yang secara teracak (*randomly occuring errors*) (Mahendra, et al., 2016).



Telah banyak dilakukan penelitian yang merancang sistem untuk deteksi dan koreksi bit *error* dengan menggunakan metode *Hamming Code*. Penelitian yang berjudul “*Perancangan Error Detection System And Error Correction System Menggunakan Metode Hamming Code Pada Pengiriman Data Text*” oleh Lubis et al (2012), yang merancang sistem deteksi dan koreksi bit *error* pada pengiriman data yang berupa teks. Penelitian ini dilakukan untuk melakukan pemeriksaan (*detection*) dan pemulihan (*correction*) *error* pada data berbentuk teks yang ditransmisikan dari komputer sumber ke komputer tujuan dengan menggunakan media transmisi *wireless*. Kemudian penelitian yang berjudul “*Deteksi dan Koreksi Multi Bit Error Dengan Partition Hamming Code*” oleh Muhajir et al (2016), merancang sistem deteksi dan koreksi bit *error* dengan menggunakan multi bit sebagai objek penelitian. Penelitian ini juga menggunakan konsep *partition Hamming Code* yang digunakan dalam proses *encode* agar mudah dalam melakukan deteksi dan koreksi *multi bit error*. Namun kelemahan dari metode ini yaitu terlalu banyak *bit* yang dikirim mengakibatkan sistem menjadi *overbit* dalam pemrosesannya. Penelitian terakhir yaitu berjudul “*Simulasi Deteksi Bit Error Menggunakan Metode Hamming Code Berbasis Web*” oleh Mahendra et al (2016), yang merancang sistem deteksi dan koreksi 1 bit *error* menggunakan web sebagai tampilan antarmuka *user* untuk mengirim data melalui web. Data yang dimasukkan hanya berupa bilangan biner saja yang telah ditentukan panjangnya 8 bit serta hanya dapat melakukan deteksi dan koreksi 1 bit *error*.

Berdasarkan beberapa penelitian yang telah dilakukan, metode *Hamming Code* dapat melakukan deteksi dan koreksi bit *error* pada data yang dikirimkan. Terdapat beberapa konsep yang diterapkan pada metode *Hamming Code* seperti menambahkan *parity bit* dalam semua blok pesan agar menjadi sebuah *codeword* baru atau membuat pola *partition bit* yang dapat memecah sebuah data menjadi lebih kecil daripada *parity Hamming Code*. Namun, dalam penelitian yang pernah dilakukan masih terdapat beberapa kekurangan dalam proses implementasi metode *Hamming Code*. Hal ini yang mendasari penulis untuk menerapkan metode *Hamming Code* dalam melakukan deteksi dan koreksi *error* dalam pengiriman data. Pada penelitian ini, penulis merancang sistem agar dapat melakukan deteksi dan koreksi *error* pada pengiriman data yang diimplementasikan pada perangkat keras yaitu *Arduino Uno*. Metode yang digunakan berupa *Hamming Code* dengan memakai konsep penambahan *parity bit Hamming Code*. Serta diimplementasikan pada sistem yang memiliki komunikasi secara serial. Penelitian ini juga memakai tipe komunikasi UART pada *Arduino Uno* yang memanfaatkan pin *tx* dan *rx* sebagai pengiriman data dan data suhu sebagai objek penelitian.

## 1.2 Rumusan masalah

Rumusan masalah dalam penelitian ini yaitu :

1. Bagaimana menerapkan metode *Hamming Code* agar sistem dapat melakukan deteksi dan koreksi bit *error* yang mengirimkan data secara serial antar mikrokontroler *Arduino* ?

2. Bagaimana hasil penerapan metode *Hamming Code* pada sistem deteksi dan koreksi bit *error* secara serial antar dua mikrokontroller Arduino ?
3. Berapa rata-rata *delay* yang dibutuhkan untuk melakukan proses *encode* dan *decode* data menggunakan metode *Hamming Code* pada *Arduino Uno*?

### 1.3 Tujuan

Pada penelitian ini, tujuan yang ingin dicapai yaitu :

1. Dapat mengimplementasikan Metode *Hamming Code* pada sistem agar dapat melakukan deteksi dan koreksi *error* pengiriman data yang dilakukan secara serial dengan berbasis UART.
2. Dapat mengetahui hasil implementasi metode *Hamming Code* yang dilakukan serta melakukan analisis pada hasil didapatkan.
3. Untuk mengetahui waktu *delay* atau waktu yang dibutuhkan sistem untuk melakukan proses *encode* dan *decode* data dengan menggunakan metode *Hamming Code*.

### 1.4 Manfaat

Dengan adanya perancangan sistem “Implementasi Deteksi dan Koreksi Error Pada Komunikasi Serial Arduino Berbasis UART Dengan Metode Hamming Code” diharapkan sistem dapat melakukan deteksi dan koreksi bit error tanpa perangkat tambahan menggunakan metode *Hamming Code*. Serta implementasi dari metode *Hamming Code* dapat diterapkan pada semua sistem yang menggunakan komunikasi serial berbasis *Arduino Uno* dan komunikasi *UART*.

### 1.5 Batasan masalah

Pada penelitian sistem “Implementasi Deteksi dan Koreksi Error Pada Komunikasi Serial Arduino Berbasis UART Dengan Metode Hamming Code” terdapat beberapa batasan penelitian yaitu:

1. Sistem hanya diimplementasikan pada mikrokontroller *Arduino* yang berbasis bahasa pemrograman C++.
2. Tampilan antarmuka untuk melakukan pengujian hanya menggunakan serial monitor pada program *Arduino IDE*.
3. Data yang diuji untuk proses deteksi dan koreksi bit error menggunakan metode *Hamming Code* memakai data suhu bertipe *integer*.
4. Sistem dapat melakukan proses *encode* data dengan metode *Hamming Code* pada data sebanyak 3 bit – 7 bit.
5. Pada sistem pengujian, sistem akan langsung melakukan proses konversi desimal ke biner sebanyak 7 bit, serta lebih difokuskan pada deteksi dan koreksi *error* pada posisi yang dimasukkan.
6. Sistem dapat mendeteksi 2 bit *error* dan hanya dapat melakukan koreksi 1 bit *error*.

## 1.6 Sistematika pembahasan

Dalam penelitian ini, penulis melakukan penelitian dengan sistematika sebagai berikut:

- |                |   |
|----------------|---|
| <b>BAB I</b>   | <b>Pendahuluan</b><br>Pada bab ini menjelaskan lebih detail tentang latar belakang permasalahan yang mendasari penelitian ini. Dijelaskan juga tentang tujuan penelitian, manfaat penelitian, serta batasan masalah yang diterapkan pada penelitian ini.  |
| <b>BAB II</b>  | <b>Landasan Kepustakaan</b><br>Pada bab ini terdiri dari 2 subbab yaitu tinjauan pustaka dan dasar teori. Tinjauan pustaka menguraikan tentang beberapa penelitian yang sudah pernah dilakukan yang berhubungan dengan masalah dalam penelitian ini, dan dasar teori yang akan menjelaskan beberapa teori dasar yang digunakan untuk menunjang penelitian yang dilakukan. |
| <b>BAB III</b> | <b>Metodologi</b><br>Bab metodologi menjelaskan tentang langkah-langkah yang dilakukan dalam penelitian. Bab ini terdiri dari studi literatur, analisis kebutuhan, perancangan dan implementasi, pengujian dan analisis, serta kesimpulan dan saran.  |
| <b>BAB IV</b>  | <b>Rekayasa Kebutuhan Sistem</b><br>Bab ini membahas tentang analisis semua kebutuhan yang dibutuhkan dan diimplementasikan secara rinci. Dimana, bab ini akan membahas kebutuhan perangkat keras dan perangkat lunak, kebutuhan fungsional dan non fungsional, serta batasan sistem.   |
| <b>BAB V</b>   | <b>Perancangan dan Implementasi</b><br>Perancangan sistem “Implementasi Deteksi dan Koreksi Error Pada Komunikasi Serial Arduino Berbasis UART Dengan Metode Hamming Code” dilakukan ketika melewati tahap analisis kebutuhan dan akan menjelaskan setiap bagian perancangan. Kemudian dilakukan tahap implementasi dari hasil perancangan untuk diketahui hasilnya.      |
| <b>BAB VI</b>  | <b>Pengujian dan Analisis</b><br>Pengujian dilakukan setelah melewati tahap Perancangan dan Implementasi untuk mengetahui hasil yang didapatkan dari tahap-tahap sebelumnya. Setelah mendapatkan hasil pengujian, maka dapat dilakukan analisis tentang hasil metode yang telah diterapkan.   |
| <b>BAB VII</b> | <b>Kesimpulan dan Saran</b><br>Kesimpulan dilakukan setelah melewati semua tahapan mulai dari Analisis Kebutuhan sampai Pengujian. Kesimpulan hasil dari implementasi yang dilakukan akan menghasilkan saran  |

untuk sistem yang dapat digunakan dalam pengembangan selanjutnya.



## BAB 2 LANDASAN KEPUSTAKAAN

Pada bab ini terdiri dari kajian pustaka dan dasar teori. yang mendukung untuk penelitian “Implementasi Deteksi dan Koreksi Error Pada Komunikasi Serial Arduino Berbasis UART Dengan Metode Hamming Code”. Kajian pustaka berisi tentang beberapa penelitian yang pernah dilakukan dan berhubungan dengan penelitian ini. Sedangkan, dasar teori menjelaskan tentang teori-teori dasar yang berhubungan dengan penelitian.

### 2.1 Tinjauan Pustaka

Tinjauan pustaka terdiri dari beberapa jurnal yang dijadikan referensi oleh penulis untuk mendukung penelitian. Beberapa penelitian sebelumnya yang terdapat dalam jurnal atau paper ini, penulis dapat mengerti tentang perkembangan penelitian yang serupa serta dapat ditemukan hasil penelitian yang relevan sehingga mendukung penelitian ini.

*Error* dapat terjadi pada saat pengiriman data dikarenakan kesalahan pada bit-bit yang dikirimkan (Mahendra, et al., 2016). Pada penelitian yang berjudul “Simulasi Deteksi Bit *Error* Menggunakan Metode *Hamming Code* Berbasis Web” peneliti merancang sebuah aplikasi simulasi yang bertujuan untuk menggambarkan bagaimana proses koreksi bit *error* pada proses pengiriman data yang berupa angka dalam bilangan biner. Penelitian ini berbasis web sebagai tampilan antarmuka *user* untuk memasukkan data yang akan dikirim. Metode *Hamming Code* yang digunakan diletakkan dalam folder *htdocs* *XAMPP* yang merupakan *database* untuk mengakses web. Terdapat dua form yang ditampilkan ke pengguna yaitu form input data *sequence* dan form pemeriksaan *error*. Form input data *sequence* akan menampilkan halaman untuk *user* memasukkan input data yang berupa data biner dan form pemeriksaan *error* yang akan menampilkan proses deteksi dan koreksi bit *error*. Penelitian ini juga menerapkan konsep *parity check bit* pada metode *Hamming Code* serta dapat melakukan deteksi dan koreksi sebanyak satu *bit error* pada data sepanjang 8 bit.

Sistem pengiriman data pada saat ini masih kurang maksimal dan sering terjadi sering kesalahan dalam proses pengirimannya (Lubis, et al., 2012). Menurut penelitian yang berjudul “ Perancangan *Error Detection System And Error Correction System* Menggunakan Metode *Hamming Code*. Pada Pengiriman Data Text” sebagian besar sistem pengiriman data sekarang ini belum mengurangi kesalahan pada pengiriman. Sehingga dirancang sebuah aplikasi *Error Detection System* dan *Error Correction System* menggunakan metode *Hamming Code*. Sistem dirancang untuk meminimalisir terjadinya kerusakan (*error*) pada saat proses transmisi data yang dilakukan secara wireless.

Bit *Error* dapat diperbaiki dengan menerapkan *error control coding* yang meliputi *Error Detection* dan *Error Correction* (Muhajir, et al., 2016). Penelitian yang berjudul “Deteksi dan Koreksi Multi Bit *Error* Dengan Partition *Hamming Code*” ini memakai konsep partition bit pesan, yang dapat membuat pesan data



dapat dipecah menjadi lebih kecil daripada konsep *parity bit* (7,4). Pesan yang dikirim dipecah menjadi sebuah *parity* dengan menambahkan *parity bit* pada setiap blok pesan yang dipecah sehingga menjadi sebuah *codeword*. Dengan menerapkan konsep ini pada metode *Hamming Code*, metode *Hamming Code* dapat melakukan koreksi *error* multi bit. Namun kelemahan dari penerapan konsep *parity bit* pesan antara lain dapat mengakibatkan bit overhead yaitu penggunaan *parity bit* yang berlebihan dan *codeword* menjadi lebih banyak. Penelitian ini juga melakukan analisis kelebihan dan kelemahan dalam memakai 2 konsep berbeda yang diterapkan dalam metode *Hamming Code* yaitu *partition bit* dan *parity bit* (7,4).

Penelitian terakhir adalah penelitian yang diusulkan oleh penulis. Penelitian ini berjudul "*Implementasi Deteksi dan Koreksi Error Pada Komunikasi Serial Arduino Berbasis UART Dengan Metode Hamming Code*". Penelitian ini akan mengembangkan penelitian-penelitian yang sebelumnya telah dilakukan. Penggunaan metode *Hamming Code* akan diterapkan pada pengiriman data yang berupa angka antar dua perangkat keras yaitu antar mikrokontroler *Arduino* dengan menggunakan data suhu. Serta menambahkan konsep *parity bit* pada metode *Hamming Code*. Pengiriman data dilakukan pada komunikasi serial antar *Arduino* menggunakan protokol *UART* yang memanfaatkan pin *tx* dan *rx*.

## 2.2 Dasar Teori

Pada bagian dasar teori, berisi referensi yang berupa teori-teori yang berhubungan dibutuhkan dalam penelitian ini. Beberapa dasar teori yang dibutuhkan akan dijelaskan pada subbab selanjutnya.

### 2.2.1 Komunikasi Digital

Komunikasi digital memiliki kemampuan untuk dapat mengontrol informasi yang dikirimkan maupun diterima yaitu dengan melakukan penyandian atau pengkodean data yang sebelum kirim ataupun diterima (Setiawan & Suryawan, 2014). Penyandian atau pengkodean data yang dimaksud adalah teknik *encoder* dan *decoder*. Teknik *encoder* akan dilakukan ketika sumber melakukan pengiriman data. Data yang dikirim akan dirubah menjadi sebuah kode dalam bentuk biner 0 dan 1. Sedangkan teknik *decoder* akan dilakukan ketika data sampai pada penerima untuk dilakukan penafsiran atau menerjemahkan data yang berupa kode dari pengirim. Model komunikasi digital ditunjukkan pada Gambar 2.1.



Gambar 2.1 Model komunikasi digital

Data yang dikirim biasanya akan melalui sistem transmisi yaitu sistem yang akan mengirim ke tujuan dari sumber. Data atau pesan yang ditransmisikan biasanya melalui sebuah saluran kabel atau nirkabel. Sistem transmisi berupa jalur

tunggal (single transmission line) atau jaringan kompleks (complex network) yang menghubungkan antara sumber dengan tujuan (Lubis, et al., 2012).

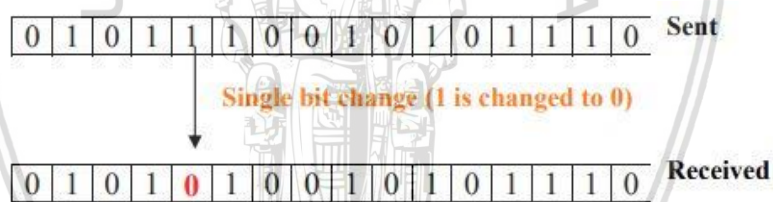
Dalam penggunaan saluran nirkabel atau kabel tidak selamanya informasi tersebut sampai dengan benar, bisa saja terjadi *error* yang menyebabkan pesan yang dikirim berbeda dengan pesan yang diterima (Muhajir, et al., 2016). Gangguan /noise dapat terjadi pada saluran komunikasi dan menyebabkan informasi yang disampaikan oleh penerima berbeda. Gangguan / noise ini akan menimbulkan kesalahan yang disebut *error*. Namun kesalahan yang disebabkan oleh gangguan/noise dapat dikurangi ke tingkat yang diinginkan ketika data rate dibatasi oleh kapasitas saluran (Hamming, 1950).

### 2.2.2 Jenis-jenis Error

Interferensi/gangguan pada lingkungan dan cacat fisik pada alat sistem komunikasi dapat menyebabkan bit *error* secara random selama proses transmisi data. Interferensi/gangguan ini dapat mengubah timing dan bentuk sinyal. Perubahan tersebut dapat mengubah informasi pada data (Kharagpur, 2008). Bahkan jika kita bisa memperkirakan kesalahan apa yang terjadi, namun kita tidak dapat mengenali kesalahan tersebut dengan mudah. Berikut adalah tipe-tipe kesalahan/*error* pada komunikasi digital :

a. *Single-bit error*

*Single-bit error* merupakan suatu kesalahan yang hanya terdiri dari satu bit unit data yang diberikan seperti yang digambarkan pada Gambar 2.2 :



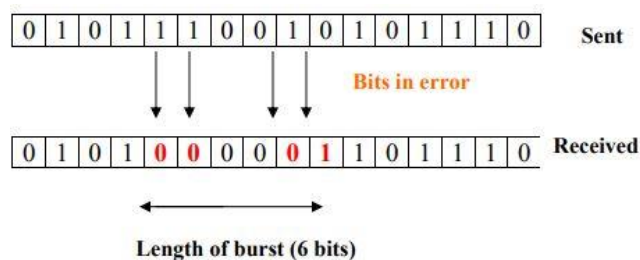
**Gambar 2.2 Model Single-bit Error**

Sumber: (Kharagpur, 2008)

Jenis kesalahan ini jarang terjadi pada pengiriman data serial, namun dapat terjadi pada pengiriman data parallel (Kharagpur, 2008).

b. *Burst Error*

*Burst error* merupakan kesalahan yang bit *error* lebih dari satu bit unit data yang telah berubah menjadi 0 ke 1 atau sebaliknya.



**Gambar 2.3 Model Burst Error**

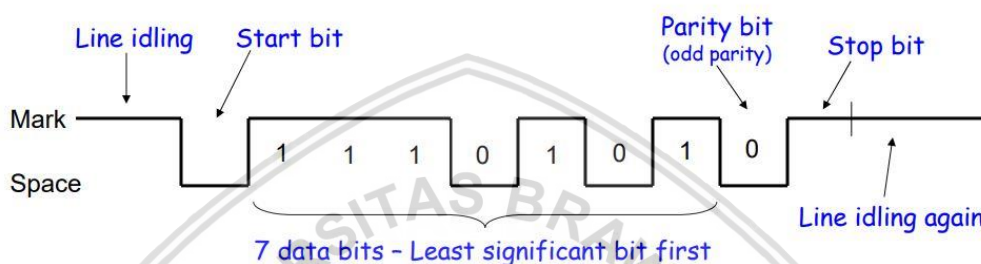
Sumber: (Kharagpur, 2008)



Berdasarkan Gambar 2.3, panjang *Burst Error* diukur dari bit *error* pertama ke bit *error* terakhir.

### 2.2.3 Komunikasi UART

UART (*Universal Asynchronous Receiver Transmitter*) merupakan salah satu jenis komunikasi serial. Komunikasi *UART* terdiri dari pin *tx* dan *rx* dalam penggunaannya. Pengiriman data yang dilakukan dengan menggunakan komunikasi ini hanya membutuhkan 1 kabel transmisi saja. Namun, jika jarak dari pengiriman data yang dilakukan terlalu jauh akan membuat paket data mengalami distorsi sehingga data yang dikirim dapat mengalami *error*.

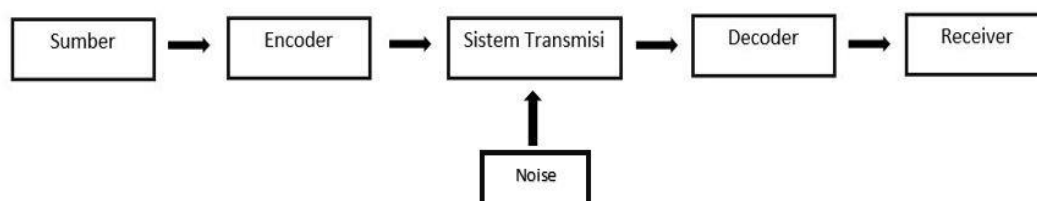


**Gambar 2.4 Pengiriman data pada komunikasi UART**

Sumber: (BYU, 2003)

Berdasarkan Gambar 2.4, komunikasi UART akan menambahkan tiga bit tambahan pada setiap pengiriman data. Tiga bit ini yaitu, *start* dan *stop* bit yang merupakan bit untuk mengawali dan menyesuaikan unit pengirim dan penerima data. *Stop bit* pada komunikasi UART akan selalu bernilai 1, jika tidak bernilai 1 maka akan terdeteksi *framing* data mengalami *error*. Sehingga *start bit* akan muncul lagi untuk melakukan pengiriman data kembali dan penerima akan menyesuaikan ulang setiap bit yang diterima. Bit yang terakhir yaitu *parity check bit*, merupakan bit untuk melakukan cek *error* data yang dikirim. *Parity check bit* ini ditambahkan dengan menggunakan metode *even/odd parity* atau dengan operasi logika XOR. Namun kelemahan dari *parity check bit* pada komunikasi UART adalah tidak dapat mendeteksi posisi *error* data dan tidak dapat melakukan koreksi *bit error*.

### 2.2.4 Error Control Coding



**Gambar 2.5 Model komunikasi digital yang terdapat noise**

Gangguan/noise biasanya dapat terjadi pada saluran sistem transmisi seperti yang digambarkan pada Gambar 2.5. *Error Control coding* berkaitan dengan deteksi dan koreksi kesalahan transmisi yang disebabkan oleh gangguan/noise di bagian sistem transmisi (Muhajir, et al., 2016). Menurut Shannon yang memperkenalkan teori dasar tentang batas komunikasi, probabilitas kesalahan data dapat terjadi dengan tingkat serendah-rendahnya pada proses pengkodean pada saluran manapun. Saat ini teori *error control coding* sudah banyak berkembang. *Error Control coding* digunakan secara luas dalam bidang komunikasi modern seperti pada radio, televisi, jaringan telepon, komputer, dan sistem komunikasi dalam ruang angkasa (Olofsson, 2005).

Gagasan umum dari teori *error control coding* yaitu dengan membiarkan *encoder* dapat menghitung bit kontrol tambahan dari informasi yang dikirim, serta untuk mengirimkan bit kontrol tersebut beserta informasinya. Jika hal tersebut dilakukan dengan tepat, maka pada sisi *decoder* dapat mendeteksi atau memperbaiki pola kesalahan yang paling mungkin terjadi (Olofsson, 2005). Dalam penerapan teknik *error control coding*, biasanya teknik ini akan digunakan untuk mengurangi probabilitas *error* yang terjadi. Hal ini dilakukan dengan mengkodekan data yang dikirim pada saluran sistem transmisi sebelum dilakukan modulasi digital. Teknik modulasi yaitu teknik yang mencampurkan sinyal menjadi satu. Biasanya sinyal yang dicampurkan adalah sinyal yang berfrekuensi tinggi dan rendah.

Dalam mengenali kesalahan yang terjadi, kita dapat membandingkan salinan isi data yang diterima dengan salinan lain pada saluran sistem transmisi. Dalam mekanisme ini blok data sumber dikirim dua kali. Metode seperti ini tidak efisien dan meningkatkan trafik pengiriman data dua atau tiga kali (Kharagpur, 2008). Oleh karena itu, terdapat strategi untuk mengatasi kesalahan dengan menambahkan banyak informasi atau bit tambahan pada *encoder* bersama dengan setiap blok data yang dikirim untuk memungkinkan penerima dapat menerima data dengan aman. Cara tersebut merupakan teknik *error control coding* yang diterapkan dalam bagian *encoder* dan *decoder* data untuk menghindari berbagai macam interferensi pada saluran sistem transmisi.

### 2.2.5 Hamming Code

Salah satu teknik *error control coding* yang sudah berkembang yaitu *Hamming Code*. *Hamming Code* merupakan teori yang ditemukan oleh Richard

Hamming pada tahun 1940. Beliau mengakui bahwa komputer membutuhkan kemampuan dalam mendeteksi dan memperbaiki kesalahan. Dalam proses transmisi, informasi dapat menjadi rusak oleh karena itu, kita mendapat sinyal di penerima yang berbeda dari sinyal aslinya (Singh, 2016). Sehingga agar sebuah perangkat dapat melakukan deteksi dan koreksi kesalahan adalah dengan menambahkan beberapa data tambahan yang mana penerima dapat menggunakan untuk memeriksa konsistensi data yang disampaikan dan untuk melakukan pemulihan bagi data yang mengalami kerusakan.

Metode *Hamming Code* merupakan metode yang bekerja dengan menyisipkan beberapa *check bit* atau *parity bit* ke dalam data. *Parity bit* ini berfungsi untuk mendeteksi bit yang *error* serta mengoreksinya. *Hamming Code* juga merupakan teknik untuk melakukan deteksi dan koreksi *single-bit error* dalam setiap blok kode pesan (*codeword*) (Hamming, 1950). *Hamming Code* dapat mendeteksi kesalahan *single-bit error* dan *burst error*. *Hamming Code* menggunakan operasi logika *Ex-OR* (*Exclusive-OR*) dalam proses deteksi dan koreksi *error*, sedangkan input dan output data dari Algoritma *Hamming Code* berupa bilangan biner (Muhajir, et al., 2016). Jumlah *check bit* yang disisipkan tergantung pada panjang data. Rumus untuk menghitung jumlah *check bit* yang disisipkan yaitu :  $2^n \geq n+1$ , dimana  $n$  adalah panjang data.

Terdapat beberapa konsep *parity bit* yang diterapkan dalam metode *Hamming Code*. Konsep yang diterapkan tergantung pada jumlah data bit yang dikirimkan. Daftar konsep *parity bit* yang diterapkan ditunjukkan pada Tabel 2.1.

**Tabel 2.1 Parity bit Hamming Code**

<i>Parity bits</i>	Jumlah bit	Data bit	Nama
2	3	1	<i>Hamming code (3,1)</i>
3	7	4	<i>Hamming code (7,4)</i>
4	15	11	<i>Hamming code (15,11)</i>
5	31	26	<i>Hamming code (31,26)</i>
6	63	57	<i>Hamming code (63,57)</i>
7	127	120	<i>Hamming code (127,120)</i>
8	255	247	<i>Hamming code (255,247)</i>

Berdasarkan Tabel 2.1, *parity bit* yang ditambahkan pada metode *Hamming Code* dapat membuat kombinasi jumlah bit. Pada penelitian “Implementasi Deteksi dan Koreksi Error Pada Komunikasi Serial Arduino Berbasis UART Dengan Metode Hamming Code” akan melakukan *encode* data sebanyak 8 bit. Sehingga membutuhkan 4 *parity bit* tambahan sesuai dengan kombinasi *parity bit* pada tabel 2.1. Hasil *encode* data akan mengirim 12 *parity bit*. Dalam melakukan *encode* data dengan menggunakan yang menggunakan konsep *parity bit* (12,8) dilakukan dengan beberapa langkah Tabel 2.2.

Tabel 2.2 Proses penambahan *parity bit*

Posisi	1	2	3	4	5	6	7	8	9	10	11	12	Operasi XOR
Posisi 1	?		1		0	0	1		1	0	1	0	$? = 1 \wedge 0 \wedge 1 \wedge 1 \wedge 1 = 0$
Posisi 2	0	?	1		0	0	1		1	0	1	0	$? = 1 \wedge 0 \wedge 1 \wedge 0 \wedge 1 = 1$
Posisi 4	0	1	1	?	0	0	1		1	0	1	0	$? = 0 \wedge 0 \wedge 1 \wedge 0 = 1$
Posisi 8	0	1	1	1	0	0	1	?	1	0	1	0	$? = 1 \wedge 0 \wedge 1 \wedge 0 = 0$
Hasil <i>Encode</i> data													011100101010

Berdasarkan Tabel 2.2, proses *encode* data dilakukan dengan langkah di bawah ini :

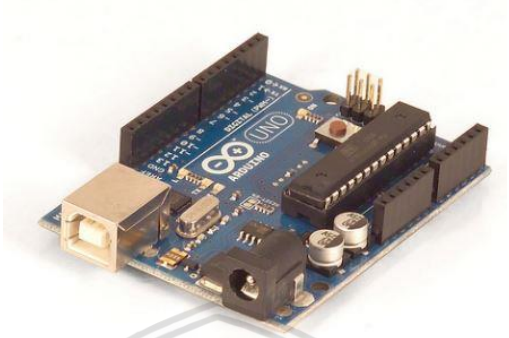
1. Menuliskan semua data bit yang akan dikirim. Contoh data yang akan diencode yaitu 10011010.
2. Berdasarkan tabel 2.2, bit yang ditulis dengan urutan 1,2,3,4,dst diberi spasi kosong dengan urutan 1,2,4,8,16,32,64,dst).
3. Kemudian melakukan cek *parity* pada setiap data bit, dengan posisi sebagai berikut:
  - Posisi 1 : melakukan operasi XOR setiap 1 bit yang berurutan ganjil dimulai dari bit pertama.
  - Posisi 2 : melakukan operasi XOR setiap 2 bit dengan urutan bit ke: 2,3,5,6,7, 10,11,14,15)
  - Posisi 4 : melakukan operasi XOR setiap 4 bit dengan urutan bit ke: 4,5,6,7,12,13,14,15,dst)
  - Posisi 8 : melakukan operasi XOR setiap 8 bit dengan urutan bit ke (8-15,24-31,40-47,dst).
4. Setelah melakukan operasi logika XOR, kemudian menambahkan *parity bit* diawal data bit yang akan dikirimkan.

Setelah melakukan langkah *encode* di atas, akan menghasilkan *codeword* data yang siap dikirim. Langkah tersebut dapat diterapkan pada semua konsep *parity bit Hamming Code*.

Selanjutnya dalam melakukan proses deteksi dan koreksi *error* data pada bagian penerima memiliki cara yang sama dengan proses *encode*. Hanya saja pada proses *decode* data yang diterima dihitung semua dengan menggunakan proses logika XOR pada setiap *parity bit* tambahan, kemudian akan menghitung pada bagian *parity bit* di posisi 1,2,4, dan 8. Jika terdapat *error* pada posisi tersebut dan lebih dari 1, dan nilai dari hasil proses pengecekan akan dikalikan nomor *parity bit* (1,2,4,dan 8) tersebut dan dijumlahkan. Hasil dari penjumlahan ini akan menentukan posisi data *error* pada bagian penerima, sehingga bagian penerima akan mengetahui data tersebut terjadi *error* atau tidak. Ketika telah dilakukan proses deteksi dan koreksi *error*, maka dapat dilakukan proses *decode* ke data asli yang dikirim.

2.2.6 Mikrokontroller *Arduino Uno*

*Arduino* merupakan mikrokontroller berbasis Atmega328. Mikrokontroller ini memiliki 14 pin digital input/output dimana 6 pin dapat digunakan sebagai output PWM dan 6 input analog.



Gambar 2.6 Mikrokontroller *Arduino Uno*

Sumber: (Arduino, 2017)

Berdasarkan Gambar 2.6, Tabel 2.3 akan menunjukkan spesifikasi dari mikrokontroller *Arduino Uno*.

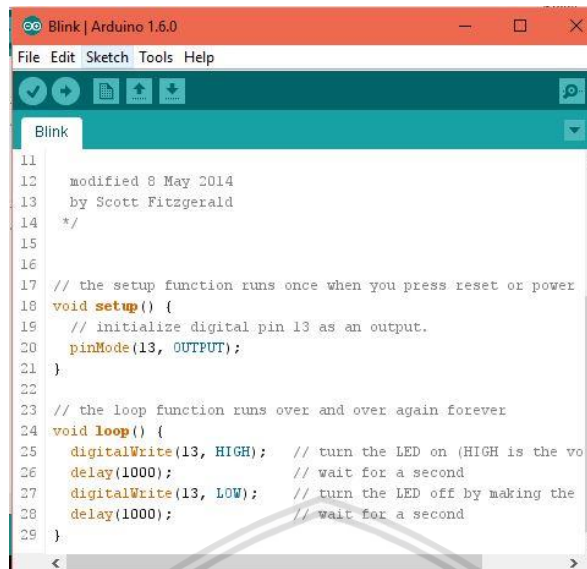
Tabel 2.3 Spesifikasi *Arduino Uno*

Tegangan	5v, maksimal 7-12v
Digital I/O Pins	14 pin
Flash Memory	32 Kb, bootloader : 0.5 Kb
SRAM	2Kb
EEPROM	1Kb
Clock Speed	16 MHz

Sumber: (Arduino, 2017)

Untuk melakukan pemrograman pada mikrokontroller *Arduino*, diperlukan *software Arduino IDE* yang ditunjukkan pada Gambar 2.7.



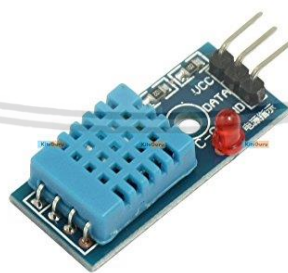


**Gambar 2.7 Arduino IDE**

*Arduino IDE* berbasis *open-source* sehingga memudahkan untuk mengembangkan penggunaan mikrokontroler *Arduino*. Dengan menuliskan kode yang berbasis bahasa C kemudian mengupload ke mikrokontroler *Arduino* melalui kabel usb yang terpasang pada komputer maka mikrokontroler *Arduino* dapat digunakan.

### 2.2.7 Sensor DHT11

Sensor DHT11 merupakan suatu sensor untuk mengukur suhu dan kelembaban. Produk ini menggunakan sensor kelembaban kapasitif dan termistor untuk mengukur udara disekitarnya, dan mengubah hasil pengukuran tersebut menjadi sinyal-sinyal digital.



**Gambar 2.8 Sensor DHT11**

Berdasarkan Gambar 2.8 spesifikasi dari sensor DHT11 ditunjukkan pada Tabel 2.4.

**Tabel 2.4 Spesifikasi Sensor DHT11**

Tegangan yang dibutuhkan	3.5V – 5.5 V
Batas pengukuran	Temperature (0°C – 50°C) Humidity (20%-90%)
Resolusi pengukuran	16 bit
Waktu respon	1 – 10 detik

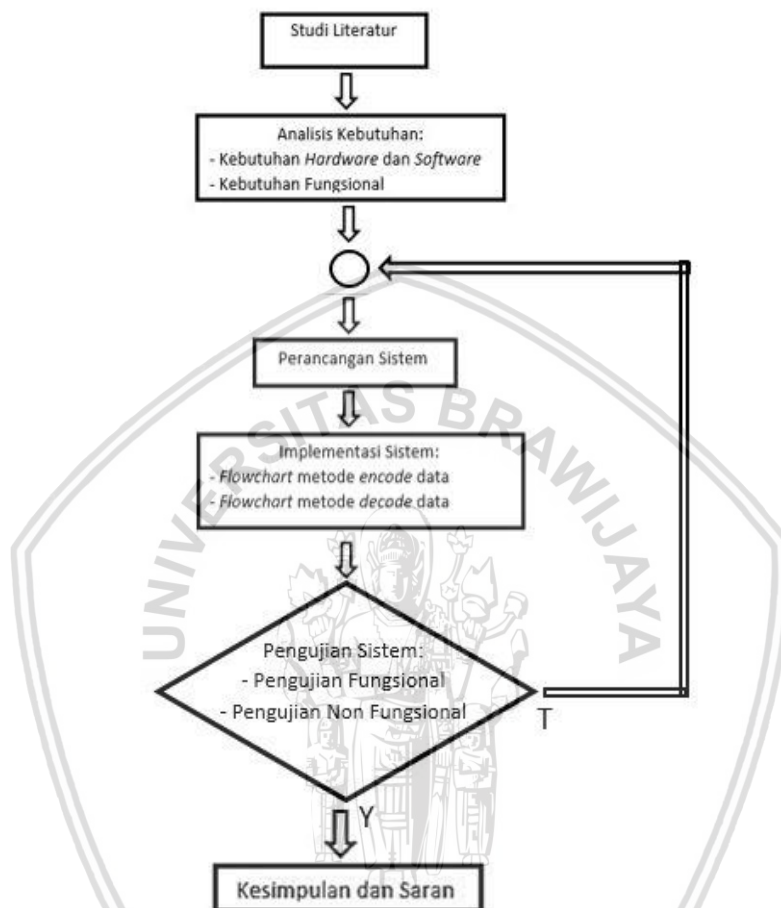
Sumber: (Liu, 2017)





## BAB 3 METODOLOGI

Metodologi penelitian merupakan suatu langkah yang berupa alur dalam melakukan penelitian. Alur dari metodologi penelitian ditunjukkan pada Gambar 3.1.



**Gambar 3.1 Diagram alir metodologi penelitian**

Berdasarkan Gambar 3.1, setiap alur pada diagram alir metodologi penelitian akan dijelaskan pada subbab-subbab selanjutnya.

### 3.1 Studi Literatur

Studi literatur digunakan untuk memperoleh informasi dan mempelajari informasi yang didapat. Informasi ini akan dijadikan sebagai referensi atau panduan dalam melakukan penelitian. Informasi yang didapat berupa literatur dari berbagai bidang ilmu yang berhubungan dengan pembuatan "Implementasi Deteksi dan Koreksi Error Pada Komunikasi Serial Arduino Berbasis UART Dengan Metode Hamming Code", diantaranya :

- Penelitian yang terkait
- *Error Control Coding*
- *Metode Hamming Code*

- Komunikasi UART

Literatur yang diperoleh dapat berupa dari buku, jurnal, artikel, atau dokumentasi suatu proyek.

### 3.2 Analisis Kebutuhan

Analisis kebutuhan bertujuan untuk mengidentifikasi kebutuhan apa saja yang dibutuhkan oleh sistem yang akan dirancang setelah melakukan studi literatur dan menentukan tujuan dari sistem. Hal ini dilakukan untuk menghindari sistem menyimpang pada tujuan yang telah ditentukan. Berikut adalah penjelasan dari analisis kebutuhan sistem “Implementasi Deteksi dan Koreksi Error Pada Komunikasi Serial Arduino Berbasis UART Dengan Metode Hamming Code”:

#### 3.2.1 Kebutuhan Perangkat Keras dan Perangkat Lunak

Analisis kebutuhan perangkat keras dan perangkat lunak dilakukan untuk mengetahui perangkat dan *software/program* apa saja yang dibutuhkan dalam melakukan penelitian ini. Perangkat keras dan perangkat lunak yang dibutuhkan antara lain:

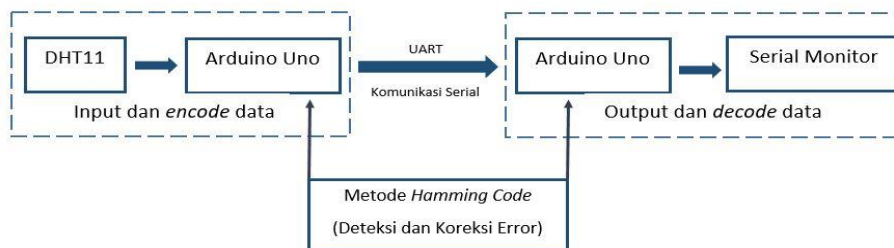
1. Laptop
2. *Arduino Uno*
3. Sensor Suhu dan Kelembaban DHT11
4. *Arduino IDE*

#### 3.2.2 Kebutuhan Fungsional

Kebutuhan fungsional merupakan suatu kebutuhan suatu proses yang dilakukan sistem dalam memenuhi tujuan dari sistem tersebut. Pada analisis kebutuhan fungsional diberikan batasan-batasan dari layanan yang dihasilkan oleh perancangan sistem. Kebutuhan fungsional sistem antara lain :

1. Sistem dapat berkomunikasi secara serial antara *Arduino Uno* dalam mengirimkan data suhu dan kelembaban.
2. *Arduino Uno* yang berperan sebagai pengirim data dapat melakukan proses *encode* data.
3. *Arduino Uno* yang berperan sebagai penerima data dapat melakukan proses *decode* data.
4. Sebelum sistem melakukan proses *encode* dan *decode*, sistem dapat mengkonversi bilangan biner sehingga dapat menjalankan metode *Hamming Code*, serta dapat melakukan deteksi dan koreksi *error bit* pada data yang dikirimkan.

### 3.3 Perancangan



**Gambar 3.2 Diagram blok perancangan sistem**

Setelah melewati kedua tahap diatas, dalam tahap ini akan mulai dirancang sistem sesuai dengan tujuan. Perancangan dilakukan ketika semua kebutuhan telah terpenuhi sesuai pada Gambar 3.2. Berdasarkan Gambar 3.2, sistem akan dirancang dengan menggunakan dua mikrokontroller yang dikomunikasikan secara serial. Komunikasi serial ini dilakukan secara *wired*. Pada *Arduino Uno* akan dirancang sistem untuk dapat melakukan deteksi suhu dan kelembaban menggunakan sensor DHT22 dan berperan sebagai pengirim data. Sedangkan pada *Arduino Uno* yang menampilkan *output* berperan sebagai penerima data. Sehingga sistem akan dirancang dengan beberapa proses yaitu :

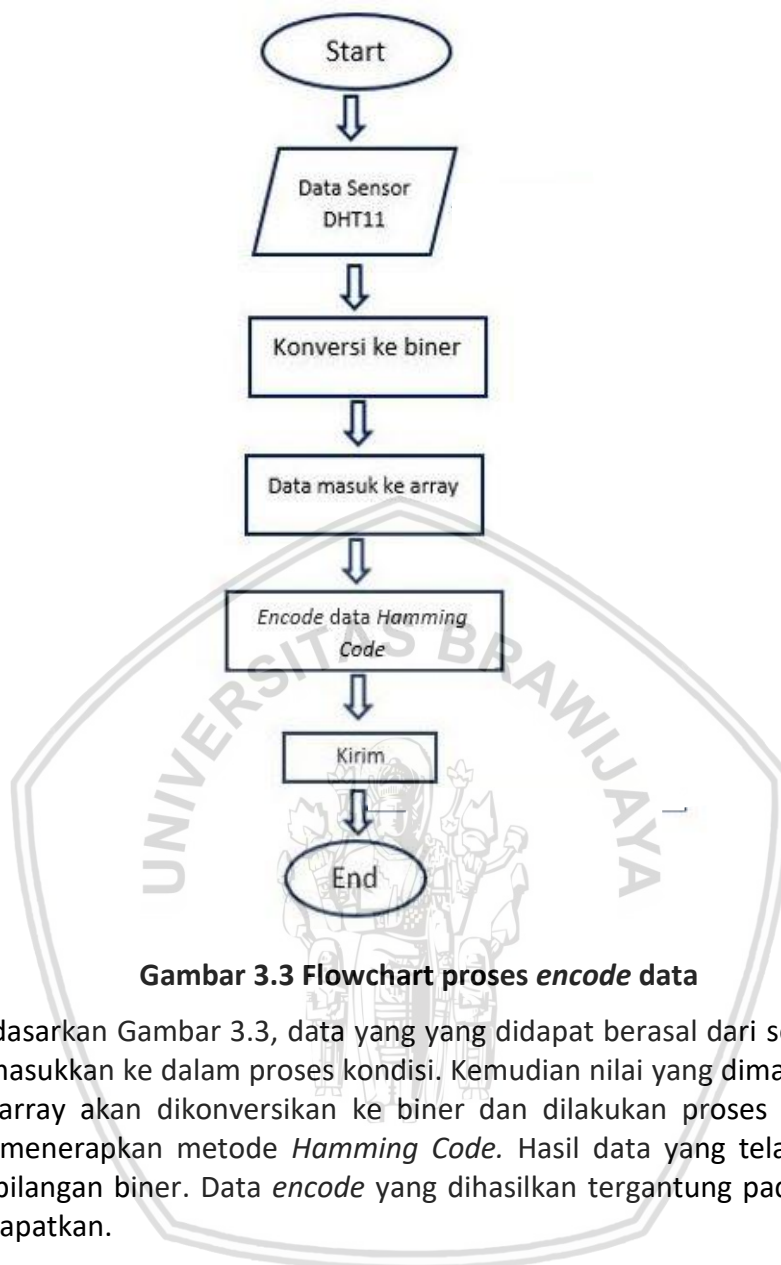
1. *Input dan encode data*. Pada proses ini akan mendapatkan data suhu dan kelembaban dari sensor DHT11. Serta akan dilakukan proses *encode*(membungkus) data yang didapatkan sebelum dikirim ke penerima.
2. *Output dan decode data*. Pada proses ini akan menerima data dari *pengirim* dan melakukan proses *decode*(membuka) data sebelum disimpan dan ditampilkan pada serial monitor.
3. Implementasi metode *Hamming Code*. Metode ini akan diterapkan pada proses *encode* dan *decode* data. Pada proses inilah yang disebut dengan *error control coding*. Sehingga data dapat dilakukan deteksi dan koreksi *error* setelah sampai dipenerima.

### 3.4 Implementasi

Setelah melakukan tahapan perancangan, selanjutnya adalah melakukan tahapan implementasi metode *Hamming Code* pada kedua proses yaitu *encode* dan *decode* data. Berikut adalah penjelasan implementasi pada bagian pengirim dan penerima.

#### 3.4.1 Implementasi Pada *Arduino Uno* (pengirim)

Pada *Arduino Uno* (pengirim), sistem akan diprogram untuk dapat melakukan proses *encode* data setelah mendapatkan data dari sensor DHT11. Flowchart proses *encode* data ditunjukkan pada Gambar 3.3

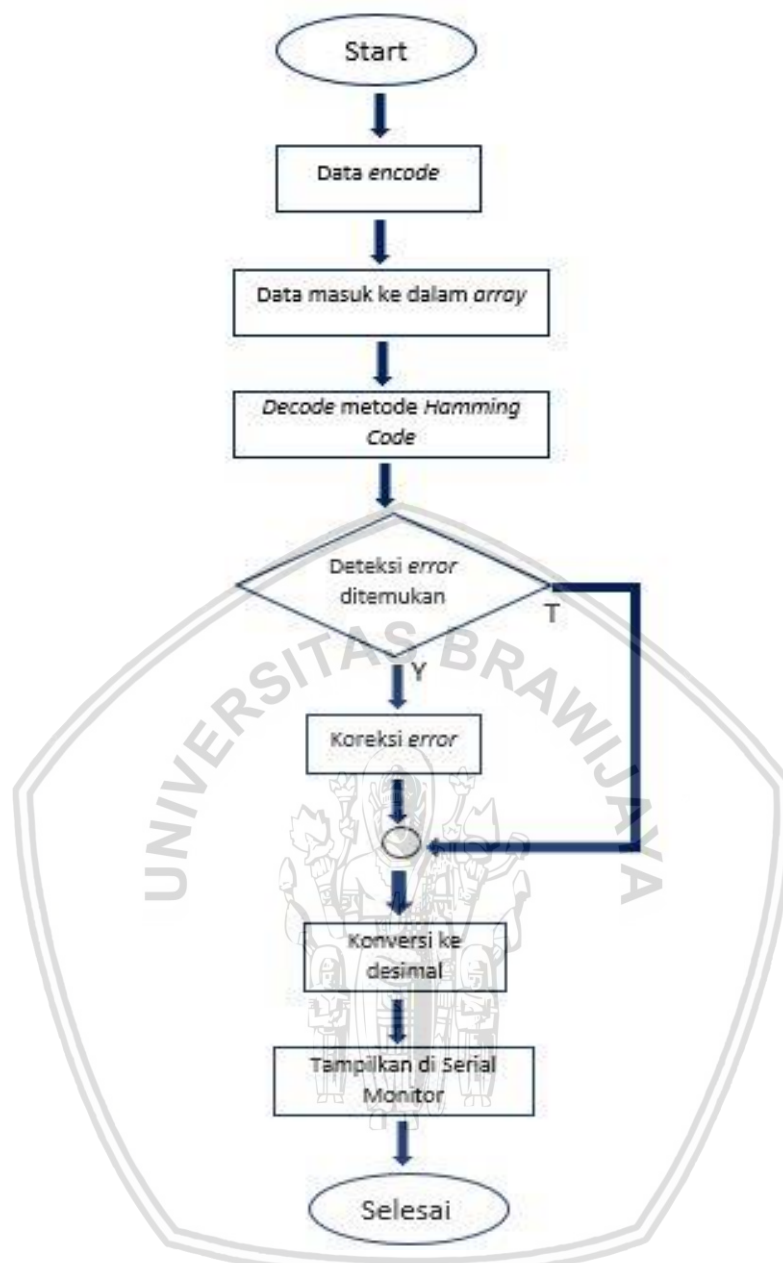


**Gambar 3.3 Flowchart proses *encode* data**

Berdasarkan Gambar 3.3, data yang yang didapat berasal dari sensor DHT11 akan dimasukkan ke dalam proses kondisi. Kemudian nilai yang dimasukkan pada sebuah array akan dikonversikan ke biner dan dilakukan proses *encode* data dengan menerapkan metode *Hamming Code*. Hasil data yang telah di *encode* berupa bilangan biner. Data *encode* yang dihasilkan tergantung pada data suhu yang didapatkan.

### 3.4.2 Implementasi Pada Penerima

Pada penerima sistem akan diprogram untuk dapat melakukan proses *decode* data setelah mendapatkan data dari pengirim. *Flowchart* proses *decode* data akan ditunjukkan pada Gambar 3.4.



**Gambar 3.4 Flowchart proses *decode* data**

Berdasarkan Gambar 3.4, data yang didapat akan langsung dimasukkan ke dalam sebuah *array*. Data ini akan dimasukkan ke dalam sebuah array. Kemudian dilakukan proses *decode* dan pengecekan *error* data dengan menerapkan metode *Hamming Code*. Pada proses pengecekan *error* data, terdapat proses untuk melakukan deteksi *error* terlebih dahulu. Jika hasil deteksi *error* ini menunjukkan adanya *error* pada data, maka akan dilakukan proses koreksi *error* untuk data dibenarkan dan di konversi ke desimal, namun jika tidak ada *error* pada data maka data akan langsung dikonversikan ke desimal dan ditampilkan.

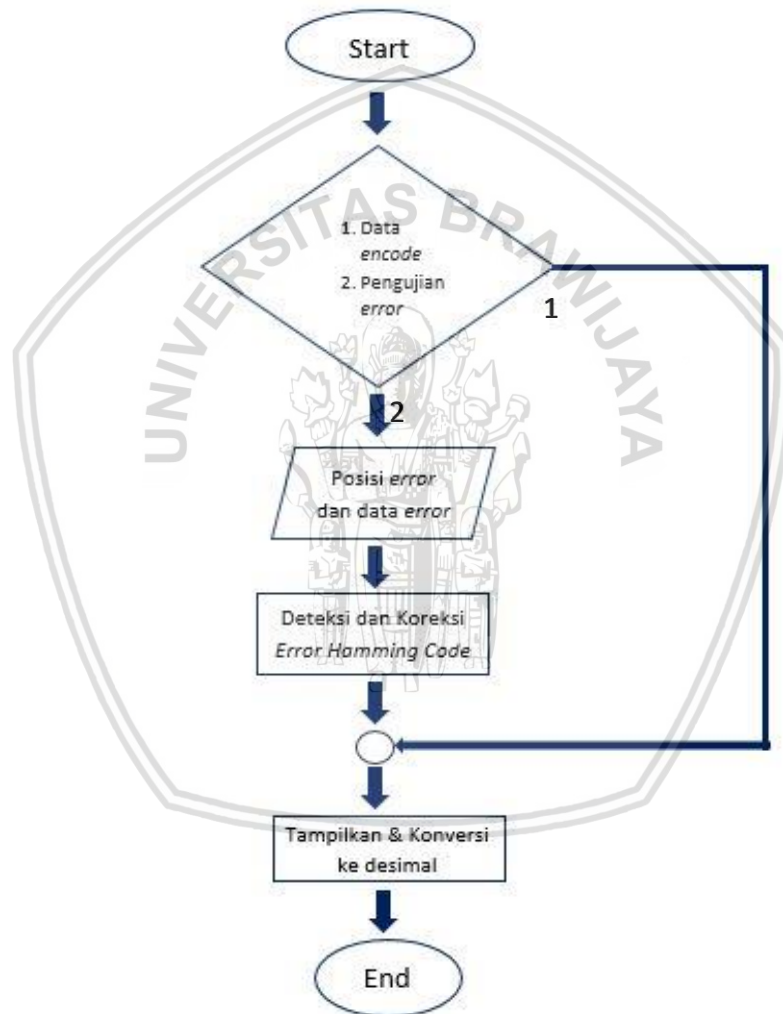
### 3.5 Pengujian dan Analisis

Pada tahap ini, dilakukan pengujian yang diambilkan dari data yang didapatkan kemudian dimasukkan kedalam sistem. Pengujian dibagi menjadi 2 yaitu :

1. Pengujian Fungsionalitas dan Non Fungsionalitas

Pengujian ini merupakan pengujian yang dilakukan untuk menguji sistem apakah telah berjalan sesuai dengan fungsi utama sistem setelah metode *Hamming Code* yang diimplementasikan. Pengujian akan dilakukan dengan menerapkan langsung pengiriman data antar *Arduino Uno* secara serial.

2. Pengujian *Error Data*



**Gambar 3.5 Flowchart Pengujian Error Data**

Berdasarkan Gambar 3.5, pada pengujian yang kedua data pada bagian penerima akan dibuat berbeda dengan data hasil *encode*. Posisi bit dan data *error* akan dimasukkan pada serial monitor untuk memberikan data yang salah. Sehingga metode *Hamming Code* yang diterapkan akan melakukan deteksi dan koreksi *error* pada data yang diterima. Pengujian ini akan dilakukan 5 kali untuk mendapatkan keakuratan dalam deteksi dan koreksi *error*.



Jika sistem yang dirancang dalam penelitian masih terdapat beberapa kesalahan yang dihasilkan dan tidak sesuai dengan tujuan sistem yang dirancang, maka alur penelitian kembali ke bagian 3 yaitu perancangan dan implementasi. Alur tetap kembali ke bagian tersebut jika sistem masih menghasilkan kesalahan yang tinggi. Kemudian analisis dilakukan setelah melakukan pengujian.

### 3.6 Kesimpulan dan Saran

Kesimpulan dan saran merupakan tahapan terakhir dalam penelitian ini. Pengambilan kesimpulan diambil berdasarkan uji coba sistem yang telah dilakukan. Tahap ini juga akan membahas tentang seberapa akurat metode *Hamming Code* yang telah diterapkan untuk melakukan deteksi dan koreksi error. Serta dapat menghasilkan saran untuk membuat sistem yang dilakukan peneliti dapat agar lebih baik kedepannya.





## BAB 4 REKAYASA KEBUTUHAN

### 4.1 Gambaran Umum Sistem

Sistem “Implementasi Deteksi dan Koreksi *Error* Pada Komunikasi Serial Arduino Berbasis *UART* Dengan Metode *Hamming Code*” dirancang dengan menggunakan mikrokontroler yaitu *Arduino Uno*. Serta dengan menggunakan sensor suhu dan kelembaban DHT11 sebagai data ujicoba yang dikirimkan pada dua mikrokontroler. Sistem dirancang dengan dua model yang berbeda yaitu:

1. Setelah dilakukan implementasi metode *Hamming Code* pada mikrokontroler *Arduino Uno*, data dari sensor DHT11 akan dikirimkan antar mikrokontroler yang sama yaitu *Arduino Uno* dan menggunakan komunikasi berbasis *UART* dalam pengirimannya.
2. Untuk melakukan pengujian deteksi dan koreksi *error* data dengan menggunakan metode *Hamming Code*, data dari sensor DHT11 yang dihubungkan dengan *Arduino Uno* sebagai pengirim akan dikirimkan ke dengan menggunakan komunikasi yang sama yaitu komunikasi *UART*.

### 4.2 Batasan Sistem

Dalam perancangan sistem “Implementasi Deteksi dan Koreksi *Error* Pada Komunikasi Serial Arduino Berbasis *UART* Dengan Metode *Hamming Code*” terdapat beberapa batasan yang diterapkan. Batasan sistem ini bertujuan agar penelitian bisa lebih fokus pada suatu tujuan dan penelitian yang dilakukan tidak keluar dari topik pembahasan dari permasalahan. Terdapat beberapa batasan sistem yang diterapkan yaitu:

1. Jenis data yang diolah dalam implementasi metode *Hamming Code* berupa tipe data suhu bertipe *integer*. *Integer* adalah tipe data pada pemrograman yang berupa bilangan bulat. Sehingga dalam penerapan metode *Hamming Code* yang hanya dapat mengolah bilangan biner, tipe data *integer* lebih cocok untuk dijadikan objek penelitian.
2. Maksimal data yang diolah dalam implementasi metode *Hamming Code* berjumlah 7 bit bilangan biner. Jumlah 7 bit ini, dapat merepresentasikan maksimal angka desimal dari pengukuran data sensor suhu dan kelembaban yaitu 100.
3. Deteksi dan koreksi *error* hanya dapat mendeteksi dan mengoreksi 1 bit *error* dalam bentuk bilangan biner pada bit dapat yang dikirimkan atau diterima.
4. Tampilan antarmuka pengujian hanya menggunakan serial monitor pada program *Arduino IDE*. Pengujian menggunakan dua serial sebagai penerima data dari pengirim dan dari *PC*.

### 4.3 Analisis Kebutuhan

Analisis kebutuhan sistem dilakukan untuk mengidentifikasi dan mempersiapkan semua kebutuhan hardware/software, fungsional atau non fungsional dalam merancang sistem “Implementasi Deteksi dan Koreksi *Error* Pada Komunikasi Serial Arduino Berbasis *UART* Dengan Metode *Hamming Code*”. Dalam melakukan analisis kebutuhan terdapat penjelasan secara rinci tentang semua kebutuhan yang dibutuhkan.

#### 4.3.1 Kebutuhan Fungsional

Melalui kebutuhan perangkat keras dan perangkat lunak, diharapkan implementasi metode *Hamming Code* yang dilakukan dapat menjalankan kebutuhan fungsional dari sistem yang dirancang. Berikut kebutuhan fungsional dalam perancangan sistem “Implementasi Deteksi dan Koreksi *Error* Pada Komunikasi Serial Arduino Berbasis *UART* Dengan Metode *Hamming Code*”.

3. Sistem dapat membaca data suhu  
*Arduino Uno* yang berperan sebagai pengirim data, dapat membaca data suhu dan kelembaban yang didapatkan dari sensor DHT11 dalam bentuk bilangan *integer* sebelum data tersebut diencode menggunakan metode *Hamming Code*.
4. Data suhu dapat ditampilkan  
Data biner hasil proses *decode* akan dirubah kembali menjadi desimal. Sebelum dirubah kembali ke bilangan desimal, terdapat beberapa posisi data tertentu yang merupakan data asli. Data pada posisi inilah yang akan diletakkan pada *array* untuk diubah kembali menjadi desimal kemudian ditampilkan pada *serial monitor*.

#### 4.3.2 Kebutuhan Non Fungsional

Kebutuhan non fungsional merupakan kebutuhan dalam menunjang berjalannya fungsionalitas sebuah sistem. Kebutuhan ini dibedakan menjadi kebutuhan performansi sistem, perangkat keras dan kebutuhan perangkat lunak.

##### 4.3.2.1 Kebutuhan Implementasi Metode *Hamming Code*

Pada implementasi metode *Hamming Code* terdapat beberapa kemampuan sistem dalam menunjang pengiriman data sensor suhu dan kelembaban. Beberapa kebutuhan tersebut antara lain:

5. Data suhu dan kelembaban diubah menjadi biner  
Sebelum data suhu dan kelembaban diencode dengan metode *Hamming Code*, data akan diubah menjadi biner terlebih dahulu.
6. Sistem dapat melakukan proses *encode*  
Sebelum data suhu dan kelembaban dikirimkan, sistem dapat melakukan proses *encode* atau pembungkusan data dengan menggunakan metode *Hamming Code*. Pada proses *encode* ini, terdapat data *parity bit* tambahan

yang akan disisipkan pada data biner. Penambahan *parity bit* mengikuti aturan rumus dari metode *Hamming Code*.

7. Sistem dapat melakukan proses *decode*  
Data yang diterima dari sisi pengirim, akan dilakukan proses decode dengan menggunakan metode *Hamming Code*. Sistem dapat mendapatkan data asli dari pengirim setelah diproses dengan menggunakan metode *Hamming Code*. Proses ini berupa deteksi dan koreksi *error* pada data yang didapatkan. Ketika pada proses *decode* terdapat bit data yang *error* maka sistem akan melakukan koreksi pada data tersebut dengan menggunakan metode *Hamming Code*, jika tidak ada yang bit data yang *error*, maka sistem akan memasukkan data proses selanjutnya.
8. Pada sistem pengujian, kita dapat memasukkan posisi *error* untuk diuji coba. Sistem pengujian ditampilkan dengan fasilitas menu pada *serial monitor*. Menu ini berupa masukan untuk posisi *error* pada data bit yang diterima oleh sisi penerima. Sehingga ketika suatu angka dimasukkan, maka angka tersebut berupa posisi pada data bit, dan akan membuat data pada posisi tersebut menjadi salah. Sehingga metode *Hamming Code* akan melakukan proses deteksi dan koreksi data sebelum data tersebut dikonversi ke desimal. Pada proses pengujian, data yang akan diencode sebanyak 7 bit dan yang didecode sebanyak 12 bit.
1. Sistem dapat melakukan komunikasi pertukaran data antar mikrokontroller dengan menggunakan protokol *UART*. Sehingga dengan implementasi metode *Hamming Code* tidak mempengaruhi berjalannya komunikasi data antar mikrokontroller.

#### 4.3.2.2 Kebutuhan Perangkat Keras

Kebutuhan perangkat keras merupakan sebuah kebutuhan pokok dalam merancang sistem “Implementasi Deteksi dan Koreksi *Error* Pada Komunikasi Serial Arduino Berbasis *UART* Dengan Metode *Hamming Code*”. Dengan perangkat keras yang dibutuhkan, dapat mendukung implementasi metode dalam melakukan pengujian dan analisis pada perangkat keras yang digunakan. Berikut adalah penjelasan tentang beberapa perangkat keras yang dibutuhkan.

1. Laptop  
Laptop / PC (*personal computer*) dibutuhkan untuk melakukan pemrograman pada mikrokontroller atau beberapa komponen lainnya agar dapat menjalankan fungsionalitas sistem.
2. *Arduino Uno*  
*Arduino Uno* banyak digunakan dan cepat diterima oleh banyak orang dikarenakan harga yang murah dibandingkan dengan *platform* lain, sangat mudah dipelajari dan digunakan dengan menggunakan dasar bahasa pemrograman C/C++. Beberapa hal yang membedakan *Arduino Uno* dengan jenis *Arduino* lain yaitu ketersediaan kabel USB dan ketersediaan Jack DC. Kemudian pada *Arduino Uno* juga terdapat kemudahan untuk koneksi kabel

*jumper* yang guna menghubungkan *Arduino Uno* dengan komponen lainnya. Pada penelitian ini, *Arduino Uno* akan dihubungkan dengan sensor DHT11 sebagai pengolah data dan media untuk implementasi metode *Hamming Code* pada *embbded system*.

### 3. Sensor DHT11

Sensor DHT11 digunakan untuk mengambil data suhu dan kelembaban yang terdapat dilingkungan sekitar. Sensor ini terdiri dari 2 bagian yaitu sensor kelembaban dan termistor yang berfungsi untuk merekam data suhu dan kelembaban. Sensor ini juga terdapat chip ADC (*Analog Digital Converter*) untuk mengubah data suhu dan kelembaban menjadi mudah dibaca dalam bentuk bilangan digital.

## 4.3.3 Kebutuhan Perangkat Lunak

Kebutuhan perangkat lunak merupakan suatu kebutuhan untuk melakukan *programming* pada perangkat keras yang digunakan. Berikut adalah penjelasan beberapa perangkat lunak yang dibutuhkan dalam merancang sistem “Implementasi Deteksi dan Koreksi Error Pada Komunikasi Serial Arduino Berbasis UART Dengan Metode Hamming Code”.

### 1. *Arduino IDE*

*Software* ini menjalankan dan mengupload baris kode yang dituliskan ke mikrokontroler *Arduino*. *Arduino IDE* terdiri dari *editor program*, *compiler* yang berfungsi pengubah bahasa pemrograman menjadi kode biner untuk mikrokontroler, dan *uploader* untuk memuat kode dari *editor program* ke memori *Arduino*.

### 2. *DHT11 Library*

*DHT11 Library* merupakan *library Arduino* dalam menggunakan beberapa fungsi. Beberapa fungsi yang dibutuhkan dalam penelitian ini yaitu fungsi untuk mendefinisikan pin sensor DHT11 dan membaca data dari sensor DHT11.

### 3. *SoftwareSerial Library*

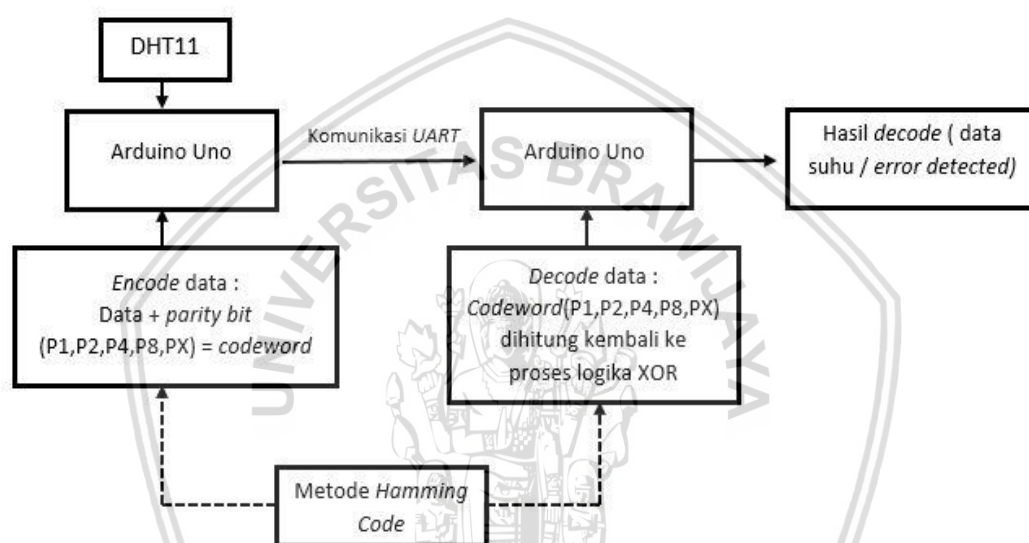
*Library* ini berfungsi untuk mengubah pin digital menjadi pin serial untuk mengirim atau menerima data. Dengan menggunakan *softwareserial*, *Arduino Uno* dapat menerima 2 jenis data yang berbeda yaitu data *input* dari *PC* dan data *input* yang dikirimkan. Dua jenis data ini akan digunakan dalam sistem pengujian.

## BAB 5 PERANCANGAN DAN IMPLEMENTASI

Bab ini membahas tentang perancangan sistem sesuai dengan analisis kebutuhan yang dilakukan dan implementasi metode *Hamming Code* pada hasil perancangan.

### 5.1 Perancangan Sistem

Pada subbab ini akan membahas tentang perancangan sistem “Implementasi Deteksi dan Koreksi *Error* Pada Komunikasi Serial Arduino Berbasis *UART* Dengan Metode *Hamming Code*”. Perancangan sistem secara umum ditunjukkan oleh diagram blok pada Gambar 5.1.



Gambar 5.1 Diagram blok perancangan sistem *encode* dan *decode* data

Berdasarkan Gambar 5.1, untuk menentukan *codeword* yang terdiri dari data asli dan *parity bit* tambahan, dilakukan perhitungan dengan rumus metode *Hamming Code* dimulai dari posisi *parity bit* ke P1,P2,P4,P8, dan Px. Kemudian *codeword* akan dikirim serta diterima kemudian dilakukan proses *decode* dengan metode *Hamming Code*. Perhitungan dilakukan kembali sama dengan proses *encode* data pada data *codeword* yang dikirim. Pada proses ini, jika nilai setiap *parity bit* (P1,P2,P4,dan P8) bernilai lebih dari 0 maka dideteksi adanya *error* pada data *codeword*. Pada variabel Px digunakan untuk mendeteksi jumlah *error* yang terjadi.

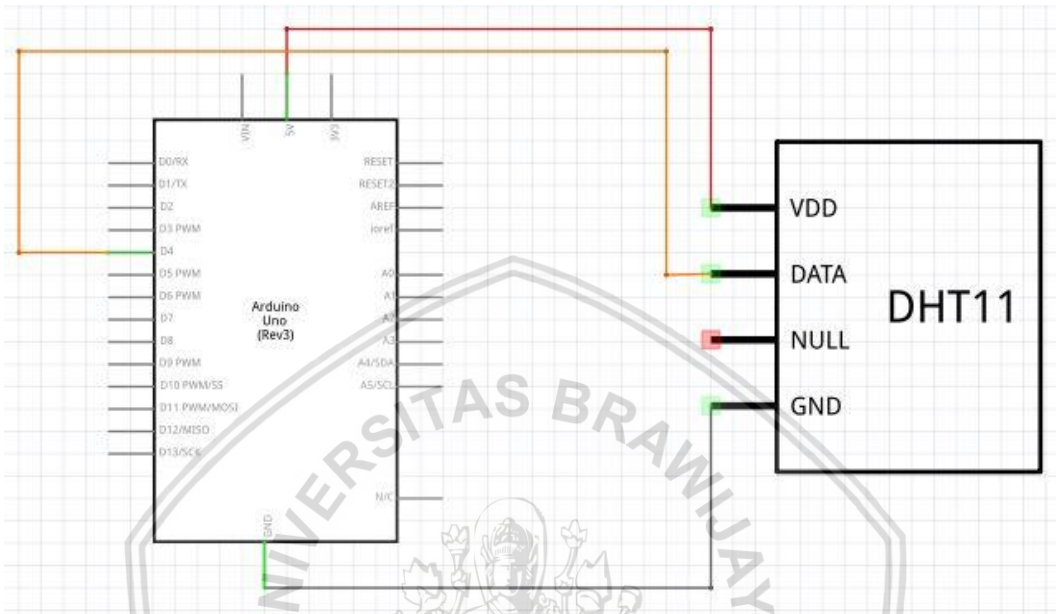
#### 5.1.1 Perancangan Perangkat Keras

Pada bab ini akan menjelaskan perancangan perangkat keras sesuai dengan kebutuhan perangkat keras yang dituliskan. Berikut adalah penjelasan detail perancangan perangkat keras dalam penelitian “Implementasi Deteksi dan Koreksi *Error* Pada Komunikasi Serial Arduino Berbasis *UART* Dengan Metode *Hamming Code*” :



5.1.1.1 Perancangan DHT11 dan Arduino Uno

Data yang akan diolah oleh metode *Hamming Code* berupa data suhu dan kelembaban yang berasal dari sensor DHT11. Skematik perancangan sensor DHT11 dan *Arduino Uno* untuk mendapatkan data suhu dan kelembaban ditunjukkan oleh Gambar 5.2.



Gambar 5.2 Skematik perancangan sensor DHT11 dan *Arduino Uno*

Berdasarkan Gambar 5.2, tabel spesifikasi pin yang digunakan akan ditunjukkan pada Tabel 5.1.

Tabel 5.1 Spesifikasi pin *Arduino Uno* dan DHT11

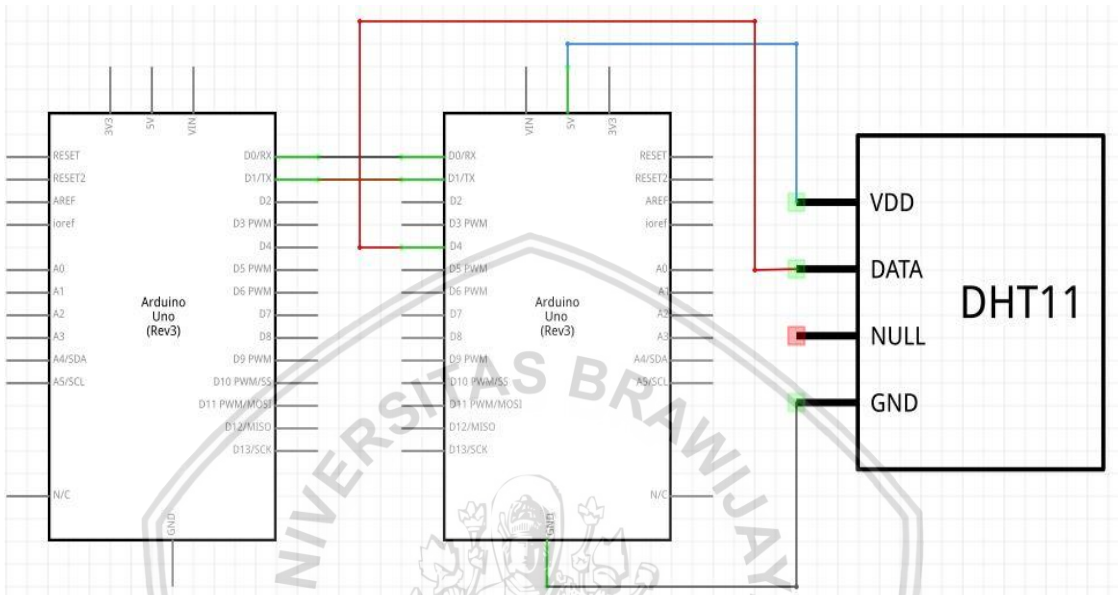
Pin Arduino Uno	Pin DHT11	Keterangan
D4	Data	Sebagai pin data antar <i>Arduino Uno</i> dan DHT11
5 V	VDD	Sebagai sumber tegangan untuk sensor DHT11
GND	GND	Sebagai <i>ground</i> untuk sensor DHT11

Berdasarkan Gambar 5.1 dan Tabel 5.1, sensor DHT11 yang terdiri dari 4 pin, dalam penelitian ini hanya menggunakan 3 pin saja. Pin *Vcc* untuk mendapatkan sumber tegangan, dihubungkan pada tegangan 5V di *Arduino Uno*. Kemudian pin data dihubungkan pada pin digital yaitu pin 4. Dan terakhir pin *ground* dihubungkan pada *GND* di *Arduino Uno*.



5.1.1.2 Perancangan Antar Mikrokontroller Arduino Uno

Pada penelitian ini, mikrokontroller *Arduino Uno* berperan sebagai suatu perangkat keras yang menjadi objek implementasi metode *Hamming Code*. Perancangan ini akan melibatkan dua *Arduino Uno* dengan menggunakan metode komunikasi *UART*. Skematik perancangan antar mikrokontroller *Arduino Uno* yang ditunjukkan pada Gambar 5.3.



Gambar 5.3 Skematik perancangan antar *Arduino Uno*

Berdasarkan Gambar 5.3, Tabel spesifikasi pin yang digunakan ditunjukkan pada Tabel 5.2.

Tabel 5.2 Spesifikasi pin antar *Arduino Uno* dan DHT11

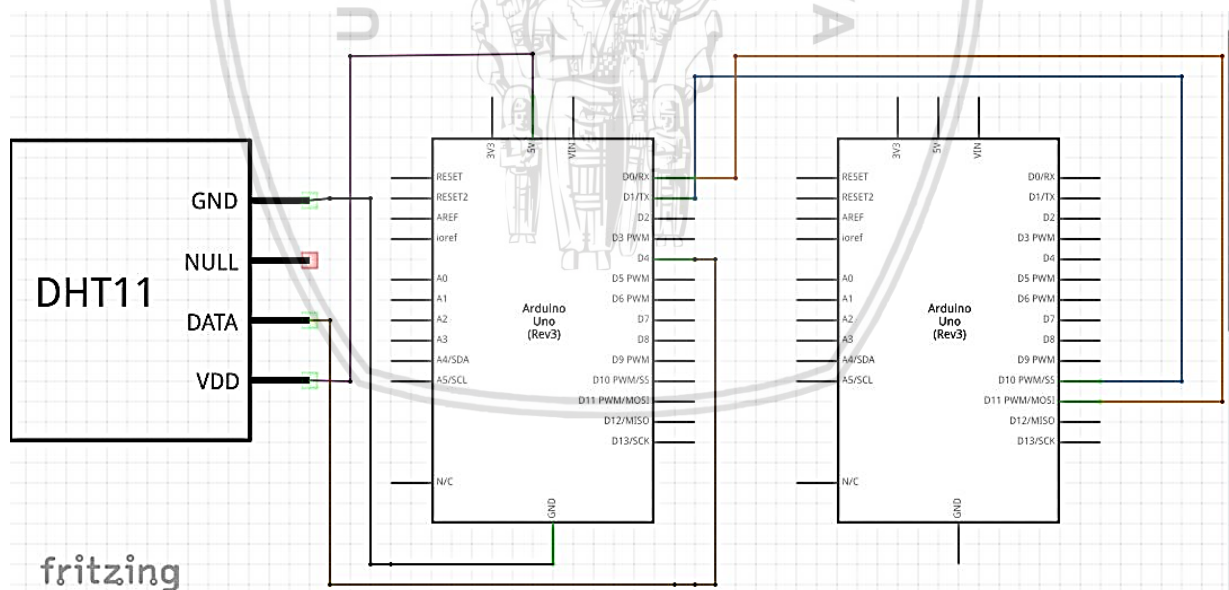
Pin Arduino Uno (penerima)	Pin Arduino Uno (pengirim)	Pin DHT11	Keterangan
D1/TX	D0/RX		Pin <i>transmitter</i> sebagai pin pengirim yang disambungkan pada pin <i>receiver</i> sebagai penerima data
D0/RX	D1/TX		Pin <i>receiver</i> sebagai pin penerima data yang disambungkan pada pin <i>transmitter</i> sebagai pengirim data

	D4	Data	Sebagai pin data antar <i>Arduino Uno</i> dan DHT11
	5 V	VDD	Sebagai sumber tegangan untuk sensor DHT11
	GND	GND	Sebagai <i>ground</i> untuk sensor DHT11

Berdasarkan Gambar 5.2 dan Tabel 5.3, kedua *Arduino Uno* ini akan melakukan komunikasi yang berupa pengiriman data secara *full-duplex*. Data yang akan dikirimkan oleh *Arduino Uno* sebagai pengirim yaitu data suhu dan kelembaban. Dimana data suhu dan kelembaban ini didapatkan dari sensor DHT11 yang dihubungkan pada pengirim data yaitu pada pin 4.

#### 5.1.1.3 Perancangan Sistem Pengujian

Perancangan antar *Arduino Uno* yang difungsikan sebagai sistem pengujian memiliki beberapa sambungan pin yang berbeda. Skematik perancangan antar *Arduino Uno* yang digambarkan pada Gambar 5.4.



Gambar 5.4 Skematik perancangan sistem pengujian

Berdasarkan Gambar 5.4, Tabel spesifikasi pin yang digunakan akan ditunjukkan pada Tabel 5.3 :

**Tabel 5.3 Spesifikasi pin sistem pengujian**

Pin <i>Arduino</i> (penerima)	Pin <i>Arduino</i> <i>Uno</i> (pengirim)	Pin DHT11	Keterangan
D11 PWM/TX	D0/RX		Pin <i>transmitter</i> sebagai pin pengirim yang disambungkan pada pin <i>receiver</i> sebagai penerima data
D10/RX	D1/TX		Pin <i>receiver</i> sebagai pin penerima data yang disambungkan pada pin <i>transmitter</i> sebagai pengirim data
	D4	Data	Sebagai pin data antar <i>Arduino Uno</i> dan DHT11
	5 V	VDD	Sebagai sumber tegangan untuk sensor DHT11
	GND	GND	Sebagai <i>ground</i> untuk sensor DHT11

Berdasarkan Gambar 5.4 dan Tabel 5.3, *Arduino Uno* yang berperan sebagai pengirim data suhu dan kelembaban dihubungkan dengan sensor DHT11. Data ini akan dikirimkan ke *Arduino Uno* yang berperan sebagai penerima data. Kedua mikrokontroler ini terdapat pin *Tx* dan *Rx* yang merupakan pin untuk berkomunikasi dengan menggunakan protokol *UART*. Pin *Tx* pada *Arduino Uno* dihubungkan pada pin *D10* yang berfungsi sebagai pin *RX* pada *Arduino Uno* penerima. Sedangkan pin *RX* *Arduino Uno* dihubungkan pada pin *D11* yang berfungsi sebagai pin *Tx* pada *Arduino Uno* penerima.

### 5.1.2 Perancangan Perangkat Lunak

Pada subbab ini menjelaskan tentang rancangan alur kerja sistem untuk melakukan kebutuhan fungsional dan nonfungsional yang digambarkan dalam bentuk *flowchart* dan beberapa konfigurasi untuk antarmuka serial sistem. Berikut adalah penjelasan rinci tentang perancangan perangkat lunak.

#### 5.1.2.1 Perancangan Antarmuka Serial

Pada penelitian ini, tampilan antarmuka untuk melihat data yang diproses menggunakan serial monitor pada program *Arduino IDE*. Perancangan serial dimulai dengan *Serial.begin(9600)* yang berarti membuka port serial dengan mengatur kecepatan pengiriman data atau *baudrate* sebesar 9600 bps(*bit per second*). Antara pengirim dan penerima menggunakan *baudrate* sebesar 9600 bps ini. Tujuannya agar urutan data yang dikirim dan diterima dapat sama.

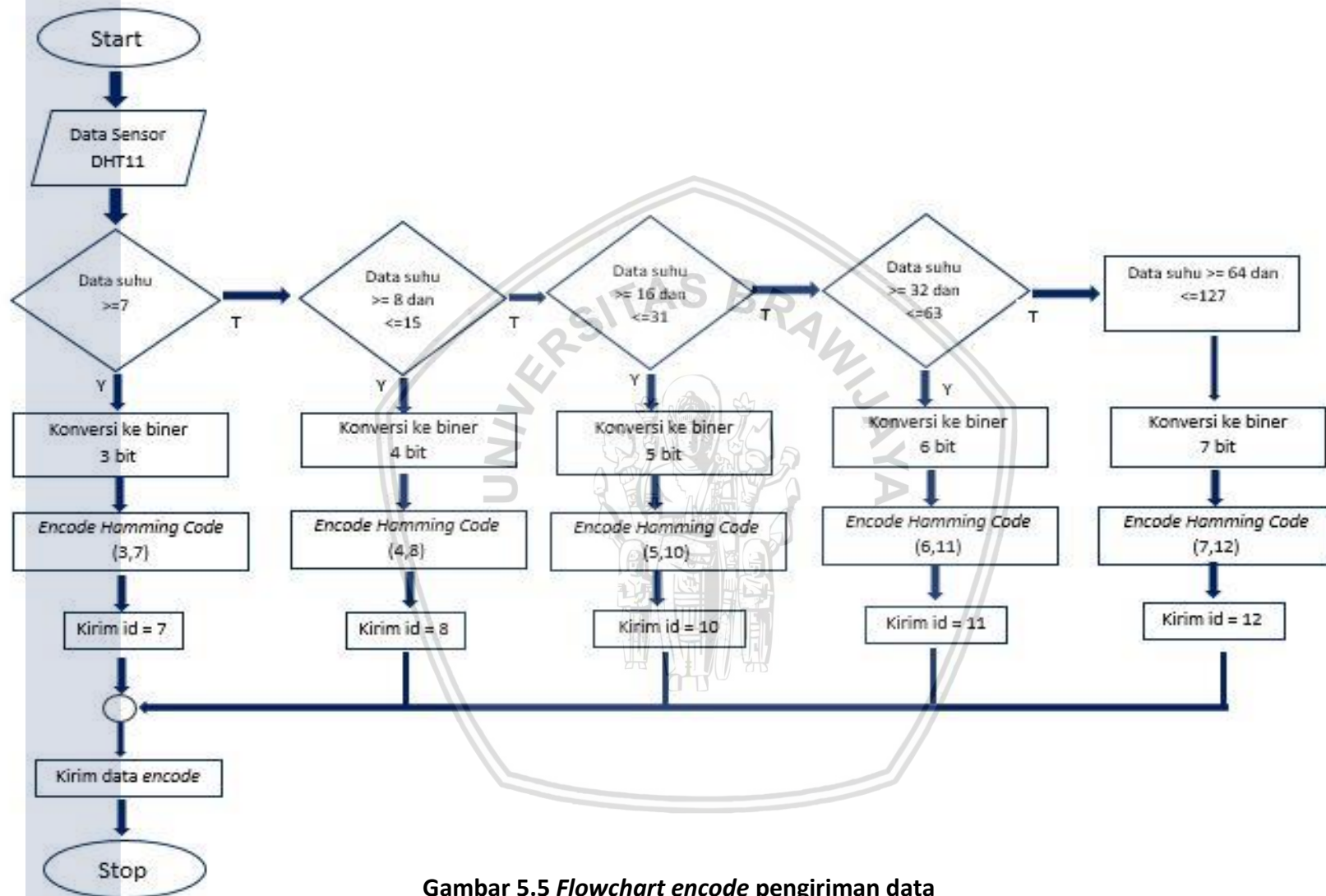
Kemudian pada sisi pengirim menggunakan kode *Serial.write* untuk menuliskan data pada komunikasi serial UART pada sisi penerima dan pengirim. Tipe variabel antar pengirim dan penerima harus disamakan. Selain itu, perancangan antarmuka serial juga menggunakan *delay* waktu sebesar 1s atau 1000ms agar data dapat diamati dan data dapat tersusun dengan benar.

#### 5.1.2.2 Perancangan Metode Hamming Code Pada Pengirim

Perancangan metode *Hamming Code* pada *Arduino Uno* akan melakukan proses *encode* data dari sensor DHT11 yaitu berupa data suhu. Pada perancangan ini akan dibedakan menjadi 2 jenis sistem, yaitu sistem untuk mengirim data suhu dan sistem untuk melakukan pengujian. Berikut adalah penjelasan lebih detail perancangan metode *Hamming Code* pada pengirim:

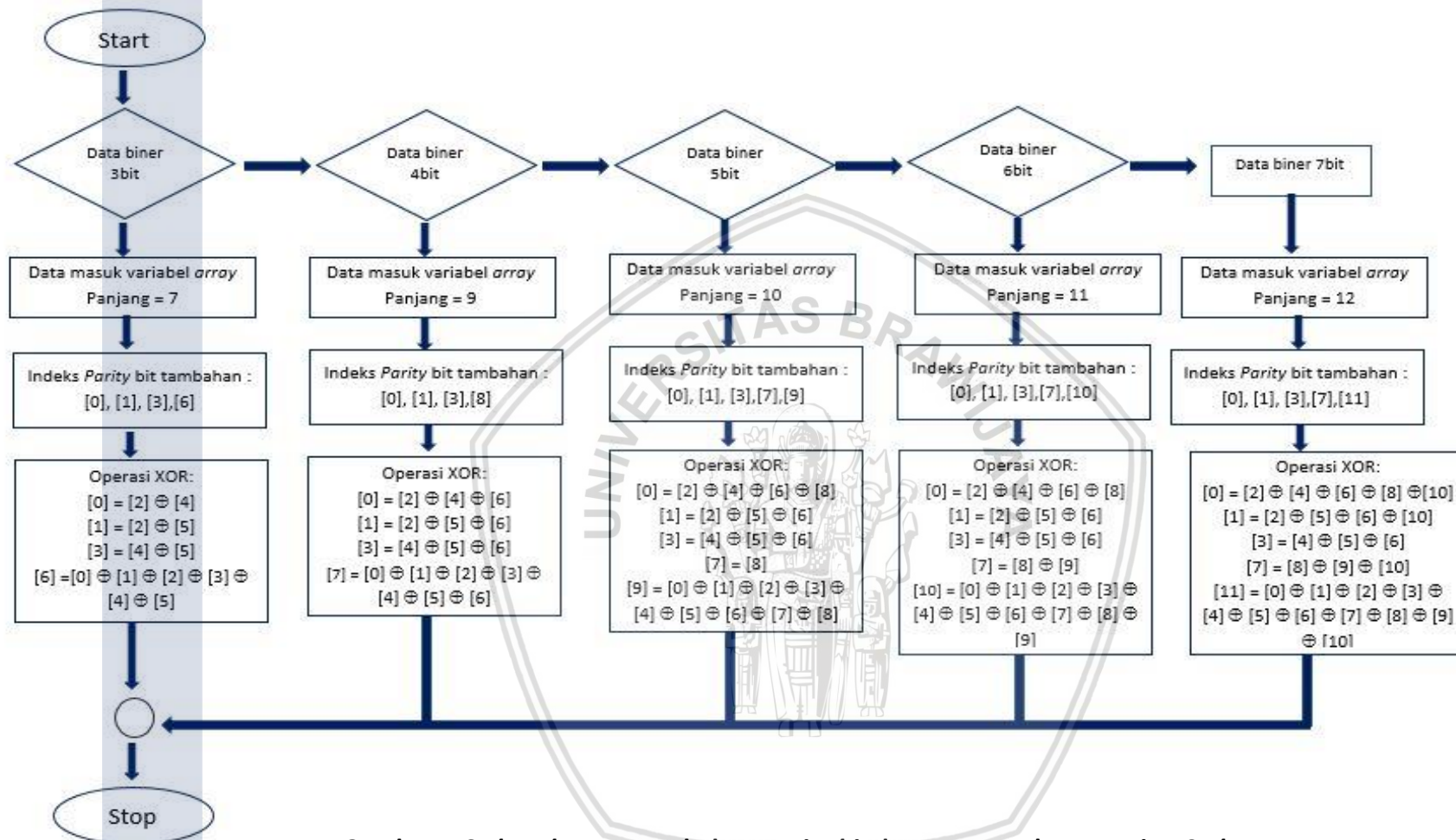
#### 5.1.2.3 Perancangan Metode Hamming Code Pada Pengiriman Data

Perancangan pada subbab ini akan ditunjukkan berupa *flowchart* pada Gambar 5.5.



Gambar 5.5 Flowchart encode pengiriman data





Gambar 5.6 Flowchart penambahan *parity bit* dengan metode *Hamming Code*

Berdasarkan Gambar 5.5, data yang akan digunakan yaitu data suhu dalam bentuk bilangan *integer*. Kemudian data ini akan dimasukkan ke sebuah kondisi untuk diubah dalam bilangan biner. Terdapat beberapa batas maksimal angka untuk menentukan banyaknya bit dalam proses konversi desimal ke biner. Hasil konversi bilangan biner ini akan dimasukkan ke dalam variabel *array* untuk dilakukan proses *encode* data dengan metode *Hamming Code*.

Sedangkan pada Gambar 5.6, jumlah bit akan mempengaruhi banyaknya *parity bit* tambahan yang akan disisipkan pada data *encode*. Indeks *array* ke 0,1,3,7, dan x merupakan indeks *parity bit* hasil dari operasi logika XOR yang dilakukan. *Parity bit* x merupakan *parity* tambahan dalam metode *Hamming Code* untuk melakukan deteksi *error* lebih dari satu bit yaitu dua bit. Letak *parity bit* ini terdapat pada *bit* terakhir data *encode*.

Operasi logika *XOR* ini menggunakan data bit biner dari data suhu tersebut. Tabel 5.4 akan menunjukkan contoh perhitungan dari operasi logika *XOR* untuk proses *encode* pada metode *Hamming Code* yang dimulai dari jumlah data sebanyak 3 *bit*.

**Tabel 5.4 Proses operasi logika xor pada metode Hamming Code (3,7)**

Data	P1	P2	D3	D4	D5	D6	Px
111	?	?	1	P	1	1	?
P1	$D3 \oplus D5 = 1 \oplus 1 = 0$						
P2	$D3 \oplus D4 = 1 \oplus 1 = 0$						
P4	$D5 \oplus D6 = 1 \oplus 1 = 0$						
Px	$P1 \oplus P2 \oplus D3 \oplus D4 \oplus D5 \oplus D6 = 0 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 = 0$						
Data encode	0010110						

**Tabel 5.5 Proses operasi logika xor pada metode Hamming Code (4,8)**

Data	P1	P2	D3	D4	D5	D6	D7	Px
1111	?	?	1	p	1	1	1	?
P1	$D3 \oplus D5 \oplus D7 = 1 \oplus 1 \oplus 1 = 1$							
P2	$D3 \oplus D6 \oplus D7 = 1 \oplus 1 \oplus 1 = 1$							
P4	$D5 \oplus D6 \oplus D7 = 1 \oplus 1 \oplus 1 = 1$							
Px	$P1 \oplus P2 \oplus D3 \oplus D4 \oplus D5 \oplus D6 \oplus D7 = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 1$							
Data encode	111111							

Pada Tabel 5.4 dan Tabel 5.5 menggunakan 4 *parity bit* tambahan yaitu pada posisi P1,P2,P4,dan Px sesuai dengan aturan metode *Hamming Code*. Sedangkan untuk bit data terletak pada posisi D3,D5,D6,D7.

**Tabel 5.6 Proses operasi logika xor pada metode *Hamming Code* (5,10)**

Data	P1	P2	D3	D4	D5	D6	D7	D8	D9	Px
11101	?	?	1	p	1	1	0	?	1	?
P1	$D3 \oplus D5 \oplus D7 \oplus D9 = 1 \oplus 1 \oplus 0 \oplus 1 = 1$									
P2	$D3 \oplus D6 \oplus D7 = 1 \oplus 1 \oplus 0 = 0$									
P4	$D5 \oplus D6 \oplus D7 = 1 \oplus 1 \oplus 0 = 0$									
P8	$D9 = 1$									
Px	$P1 \oplus P2 \oplus D3 \oplus D4 \oplus D5 \oplus D6 \oplus D7 \oplus D8 \oplus D9 = 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 1$									
Data encode	1010110110									

**Tabel 5.7 Proses operasi logika xor pada metode *Hamming Code* (6,11)**

Data	P1	P2	D3	D4	D5	D6	D7	D8	D9	D10	Px
100000	?	?	1	p	0	0	0	?	0	0	?
P1	$D3 \oplus D5 \oplus D7 \oplus D9 = 1 \oplus 0 \oplus 0 \oplus 0 = 1$										
P2	$D3 \oplus D6 \oplus D7 \oplus D10 = 1 \oplus 0 \oplus 0 \oplus 0 = 1$										
P4	$D5 \oplus D6 \oplus D7 = 1 \oplus 0 \oplus 0 = 1$										
P8	$D9 \oplus D10 = 0 \oplus 0 = 0$										
Px	$P1 \oplus P2 \oplus D3 \oplus D4 \oplus D5 \oplus D6 \oplus D7 \oplus D8 \oplus D9 \oplus D10 = 1 \oplus 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 \oplus 0 \oplus 0 \oplus 0 \oplus 0 = 1$										
Data encode	1110000001										

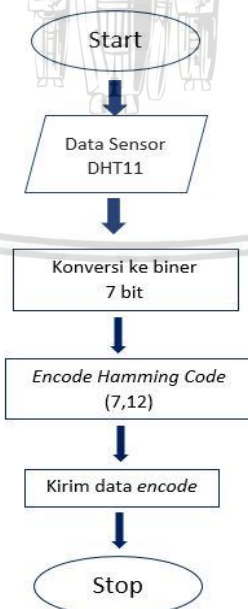
**Tabel 5.8 Proses operasi logika xor pada metode *Hamming Code* (7,12)**

Data	P1	P2	D3	D4	D5	D6	D7	D8	D9	D10	D11	Px
1010010	?	?	1	?	0	1	0	?	0	1	0	?
P1	$D3 \oplus D5 \oplus D7 \oplus D9 \oplus D11 = 1 \oplus 0 \oplus 0 \oplus 0 \oplus 0 = 1$											
P2	$D3 \oplus D6 \oplus D7 \oplus D10 \oplus D11 = 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 1$											
P4	$D5 \oplus D6 \oplus D7 = 0 \oplus 1 \oplus 0 = 1$											
P8	$D9 \oplus D10 \oplus D11 = 0 \oplus 1 \oplus 0 = 1$											
Px	$P1 \oplus P2 \oplus D3 \oplus D4 \oplus D5 \oplus D6 \oplus D7 \oplus D8 \oplus D9 \oplus D10 = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 1$											
Data encode	111101010101											

Pada Tabel 5.6 – Tabel 5.8, *parity bit* yang ditambahkan berjumlah 5 yaitu pada posisi P1,P2,P4,P8, dan Px. Pada *parity bit* ke P1,P2,P4 dan P8 dapat merepresentasikan posisi data sampai dengan jumlah data yang dikirim.

#### 5.1.2.4 Perancangan Metode Hamming Code Pada Sistem Pengujian Data Error

Pada sistem pengujian data *error* menggunakan *Arduino Uno* yang berperan sebagai pengirim. Perancangan metode *Hamming Code* pada sistem pengujian akan digambarkan pada Gambar 5.7.



**Gambar 5.7 Flowchart perancangan metode *Hamming Code* pada sistem pengujian bagian pengirim**

Berdasarkan Gambar 5.6, Sistem pengujian ini dirancang dengan data yang berbeda pada sistem pengirim sebelumnya. Sistem pengujian ini, lebih difokuskan pada penyisipan bit *error* di bagian penerima. Data yang didapatkan yaitu data suhu, akan dikonversikan ke dalam bilangan biner sebanyak 7 bit. Sehingga data suhu ini tidak dimasukkan ke dalam kondisi, dan langsung dilakukan proses konversi biner 7 bit. Hal ini dilakukan, berdasarkan studi literatur tentang pengiriman data dengan komunikasi *UART* yang mengirimkan data sebanyak 7 bit dalam sekali pengiriman.

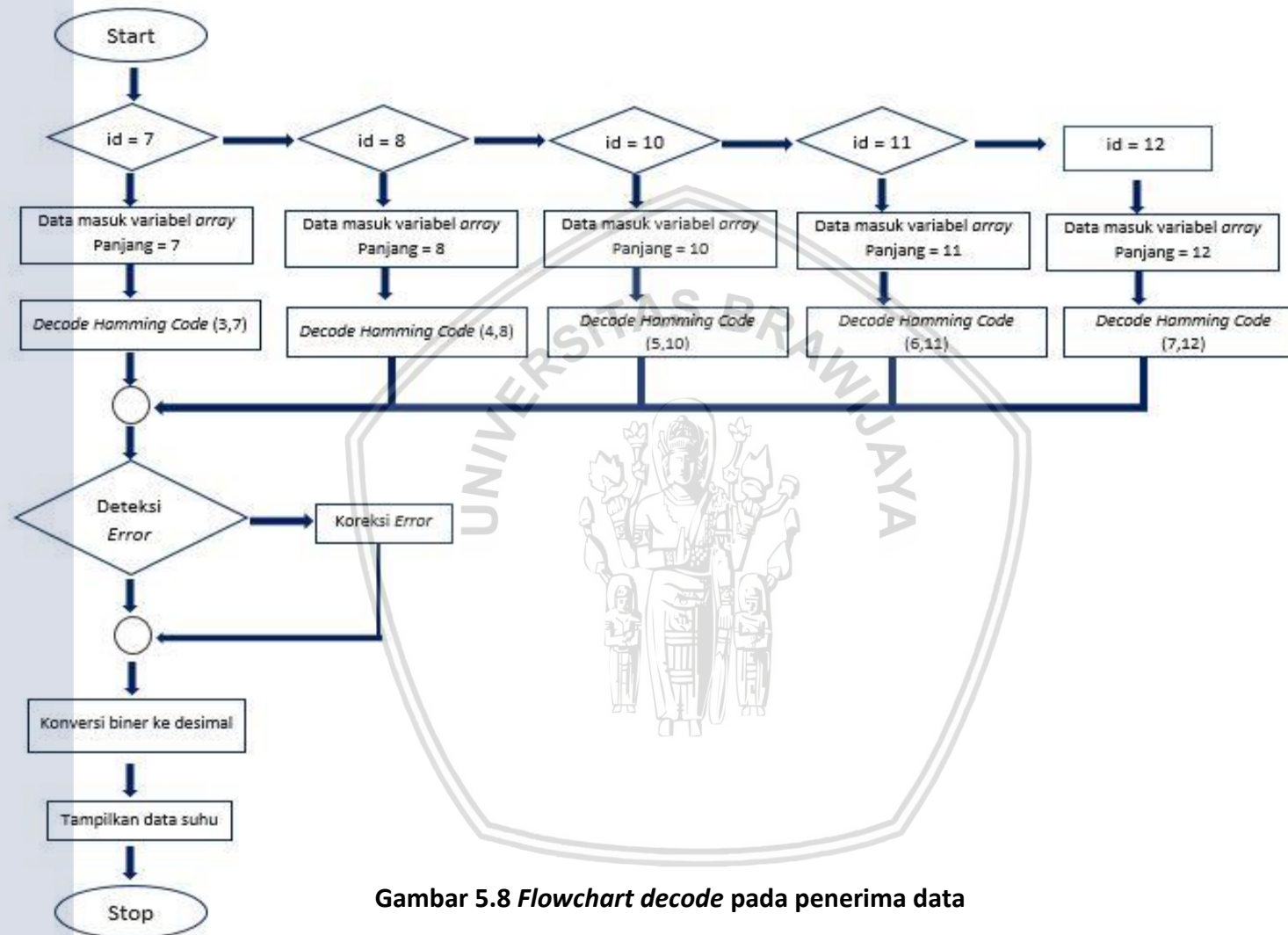
#### 5.1.2.5 Perancangan Pada Sistem Pengujian

Pada perancangan sistem pengujian, data *encode* yang telah disisipi *parity bit* tambahan dikirimkan dalam bentuk bilangan biner dengan menggunakan komunikasi *UART*. Sistem ini akan dibedakan menjadi 2 jenis perancangan, yaitu perancangan pada penerima data dan perancangan sistem pengujian.

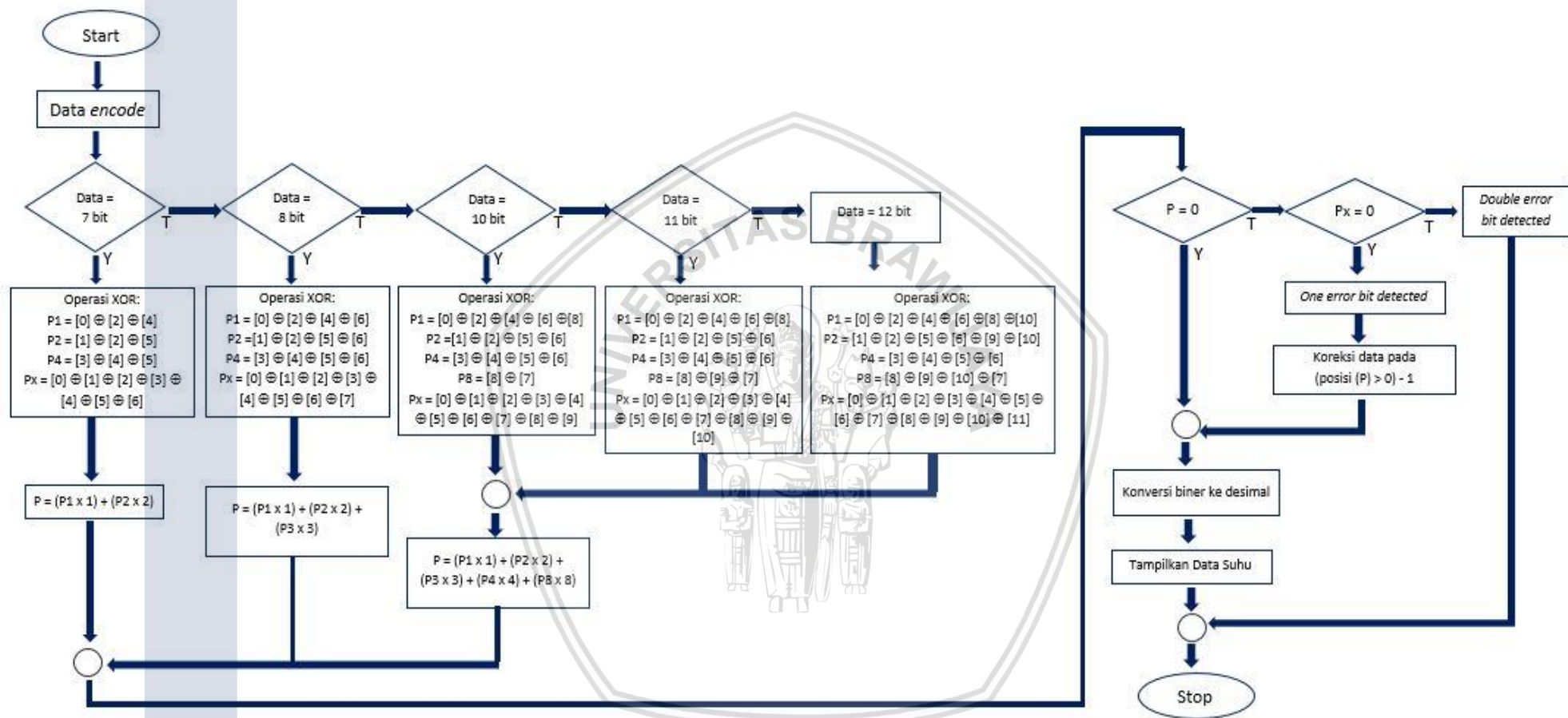
Terdapat 2 jenis data yang dikirim yaitu data *id* dan data *encode*. Data *id* ini berupa data panjang *array* yang akan menentukan proses *decode* pada bagian penerima dan data *encode* yang akan ditampilkan.

Data *encode* yang diterima akan masuk ke dalam kondisi untuk *didecode* dengan menggunakan logika XOR. Terdapat variabel P1,P2,P4,P8 yang menampung hasil operasi logika XOR atau bisa juga disebut variabel *parity bit* tambahan dan variabel P yaitu posisi yang akan menampung hasil perkalian dan penjumlahan dari variabel P1,P2,P4,P8. Variabel P akan bernilai lebih dari 0, jika terdapat data yang salah atau mengalami *error*, dan bernilai 0 yang berarti data tersebut tidak ada yang *error*. Jika nilai P lebih dari 0 maka akan dibawa ke kondisi koreksi data *error*. *Flowchart* perancangan sistem pada bagian penerima data yang ditunjukkan pada Gambar 5.8.



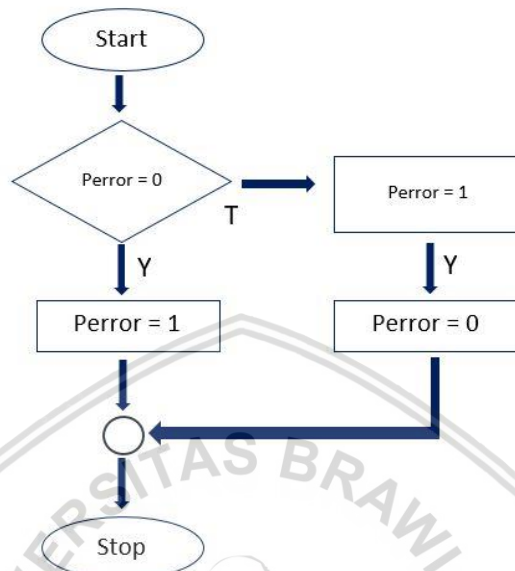


Gambar 5.8 Flowchart decode pada penerima data



Gambar 5.9 Flowchart deteksi *error* pada proses *decode* data dengan metode *Hamming Code*

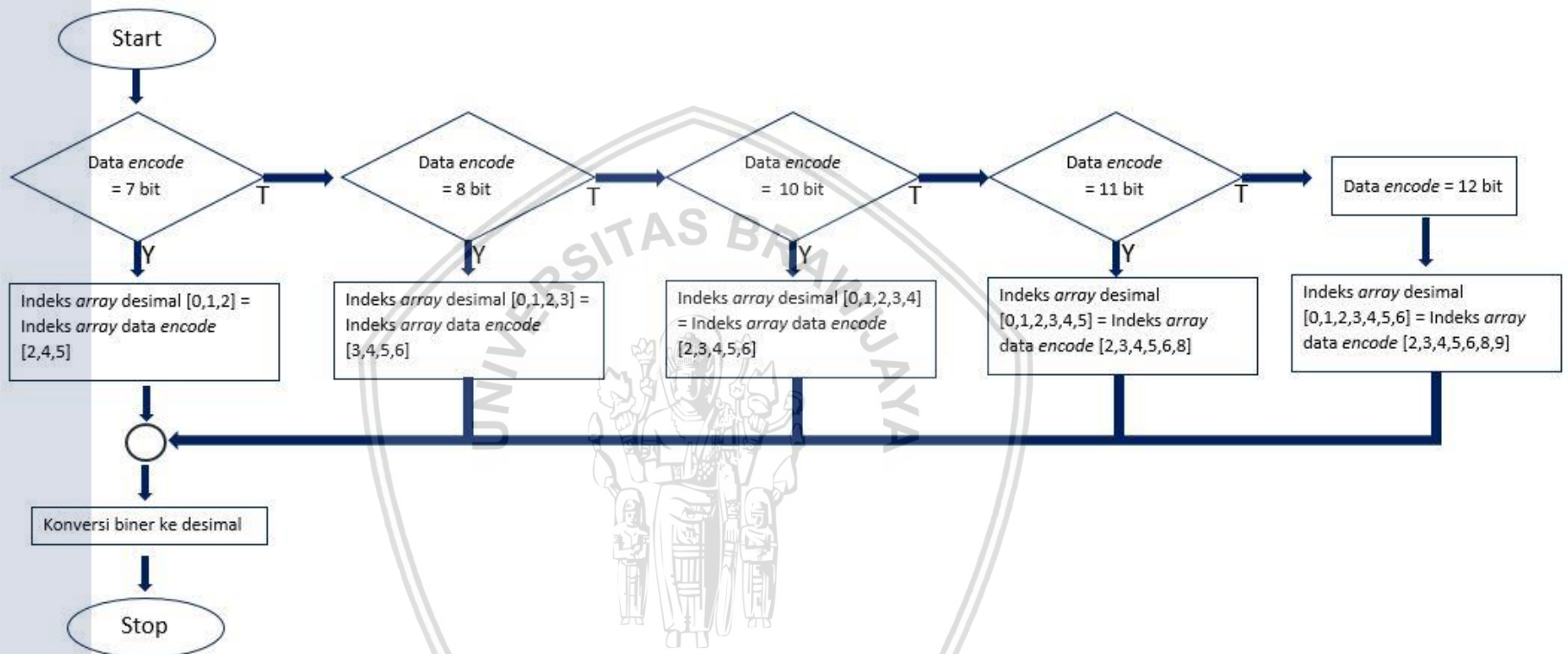
Pada Gambar 5.9, Dimana, nilai P adalah posisi dimana data tersebut mengalami *error*. Dikarenakan indeks *array* dimulai dari urutan ke 0, maka nilai P tersebut akan dikurangi 1. Setelah memasuki kondisi tersebut, maka data yang mengalami *error* akan dikoreksi dengan *flowchart* yang ditunjukkan pada Gambar 5.10.



**Gambar 5.10 Flowchart koreksi error**

Berdasarkan Gambar 5.10, Perror merupakan variabel yang menampung nilai dari posisi data *error*. Jika nilai tersebut 0 maka akan merubah data menjadi 1, jika nilai tersebut 1 maka akan merubah data menjadi 0.

Kemudian data *encode* yang telah dimasukkan ke dalam proses cek *error* metode *hamming code* akan dimasukkan lagi nilainya ke dalam indeks variabel *array* desimal untuk menampilkan data suhu. Indeks *array parity bit* tambahan yaitu indeks ke 0,1,3,dan 7 tidak dimasukkan pada variabel *array* desimal. Pada Gambar 5.10 akan menunjukkan perancangan perhitungan dari cek *error* data dengan menggunakan metode *Hamming Code*. Setelah dilakukan kondisi tersebut, maka data *encode* akan dikonversi ke desimal dengan *flowchart* pada Gambar 5.11.



Gambar 5.11 Flowchart konversi desimal ke biner

**Tabel 5.9 Perhitungan cek *error* data dengan *Hamming Code* (3,7)**

Data	P1	P2	D3	D4	D5	D6	Px
0010110	0	0	1	0	1	1	0
P1	$P1 \oplus D3 \oplus D5 = 0 \oplus 1 \oplus 1 = 0$						
P2	$P2 \oplus D3 \oplus D4 = 0 \oplus 1 \oplus 1 = 0$						
P4	$P4 \oplus D5 \oplus D6 = 0 \oplus 1 \oplus 1 = 0$						
Cek <i>error</i>	$(1 \times P1) + (2 \times P2) + (4 \times P4) = (1 \times 0) + (2 \times 0) + (4 \times 0) = 0$						
Data <i>encode</i>	0010110 = <i>no error</i>						

**Tabel 5.10 Perhitungan cek *error* data dengan *Hamming Code* (4,8)**

Data	P1	P2	D3	P4	D5	D6	D7	Px
1111111	1	1	1	1	1	1	1	1
P1	$P1 \oplus D3 \oplus D5 \oplus D7 = 1 \oplus 1 \oplus 1 \oplus 1 = 0$							
P2	$P2 \oplus D3 \oplus D6 \oplus D7 = 1 \oplus 1 \oplus 1 \oplus 1 = 0$							
P4	$P4 \oplus D5 \oplus D6 \oplus D7 = 1 \oplus 1 \oplus 1 \oplus 1 = 0$							
Cek <i>error</i>	$(1 \times P1) + (2 \times P2) + (4 \times P4) = (1 \times 0) + (2 \times 0) + (4 \times 0) = 0$							
Data <i>encode</i>	1111111 = <i>no error</i>							

Pada Tabel 5.9 dan Tabel 5.10, proses cek *error* dengan menggunakan metode *Hamming Code* melalui operasi logika *XOR* pada 4 *parity bit* tambahan yaitu pada P1, P2, P4, dan Px yang terletak pada posisi terakhir.

**Tabel 5.11 Perhitungan cek *error* data dengan *Hamming Code* (5,10)**

Data	P1	P2	D3	P4	D5	D6	D7	P8	D9	Px
1010110110	1	0	1	0	1	1	0	1	1	0
P1	$P1 \oplus D3 \oplus D5 \oplus D7 \oplus D9 = 1 \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 0$									
P2	$P2 \oplus D3 \oplus D6 \oplus D7 = 0 \oplus 1 \oplus 1 \oplus 0 = 0$									
P4	$P4 \oplus D5 \oplus D6 \oplus D7 = 0 \oplus 1 \oplus 1 \oplus 0 = 0$									
P8	$D8 \oplus D9 = 1 \oplus 1 = 0$									
Cek <i>Error</i>	$(1 \times P1) + (2 \times P2) + (4 \times P4) + (8 \times P8) = (1 \times 0) + (2 \times 0) + (4 \times 0) + (8 \times 0) = 0$									
Data <i>encode</i>	1010110110 = <i>no error</i>									



**Tabel 5.12 Perhitungan cek error data dengan Hamming Code (6,11)**

Data	P1	P2	D3	P4	D5	D6	D7	P8	D9	D10	Px
1110000001	1	1	1	0	0	0	0	0	0	0	1
P1	$P1 \oplus D3 \oplus D5 \oplus D7 \oplus D9 = 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 = 0$										
P2	$P2 \oplus D3 \oplus D6 \oplus D7 \oplus D10 = 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 = 0$										
P4	$P4 \oplus D5 \oplus D6 \oplus D7 = 1 \oplus 1 \oplus 0 \oplus 0 = 0$										
P8	$P8 \oplus D9 \oplus D10 = 0 \oplus 0 \oplus 0 = 0$										
Cek Error	$(1 \times P1) + (2 \times P2) + (4 \times P4) + (8 \times P8) = (1 \times 0) + (2 \times 0) + (4 \times 0) + (8 \times 0) = 0$										
Data encode	1110000001 = no error										

**Tabel 5.13 Perhitungan cek error data dengan Hamming Code (7,12)**

Data	P1	P2	D3	P4	D5	D6	D7	P8	D9	D10	D11	Px
111101 010101	1	1	1	1	0	1	0	1	0	1	0	1
P1	$P1 \oplus D3 \oplus D5 \oplus D7 \oplus D9 \oplus D11 = 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 \oplus 0 = 0$											
P2	$P2 \oplus D3 \oplus D6 \oplus D7 \oplus D10 \oplus D11 = 1 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 0$											
P4	$P4 \oplus D5 \oplus D6 \oplus D7 = 1 \oplus 0 \oplus 1 \oplus 0 = 0$											
P8	$P8 \oplus D9 \oplus D10 \oplus D11 = 1 \oplus 0 \oplus 1 \oplus 0 = 0$											
Cek error	$(1 \times P1) + (2 \times P2) + (4 \times P4) + (8 \times P8) = (1 \times 0) + (2 \times 0) + (4 \times 0) + (8 \times 0) = 0$											
Data encode	111101010101											

Berdasarkan Tabel 5.11 – Tabel 5.13, cek error menggunakan 5 parity bit tambahan yaitu pada posisi P1,P2,P4,P8,dan Px. Hasil nilai dari parity bit tersebut didapatkan dari operasi logika XOR antar posisi data aslinya sesuai dengan aturan Hamming Code. Parity bit tambahan akan bernilai 0, jika tidak ada data yang salah atau error. Kemudian hasil nilai parity bit ini akan dikalikan dengan posisinya, kemudian setiap nilai perkalian dengan posisinya ini akan dijumlahkan dan menentukan posisi data yang salah atau mengalami error.

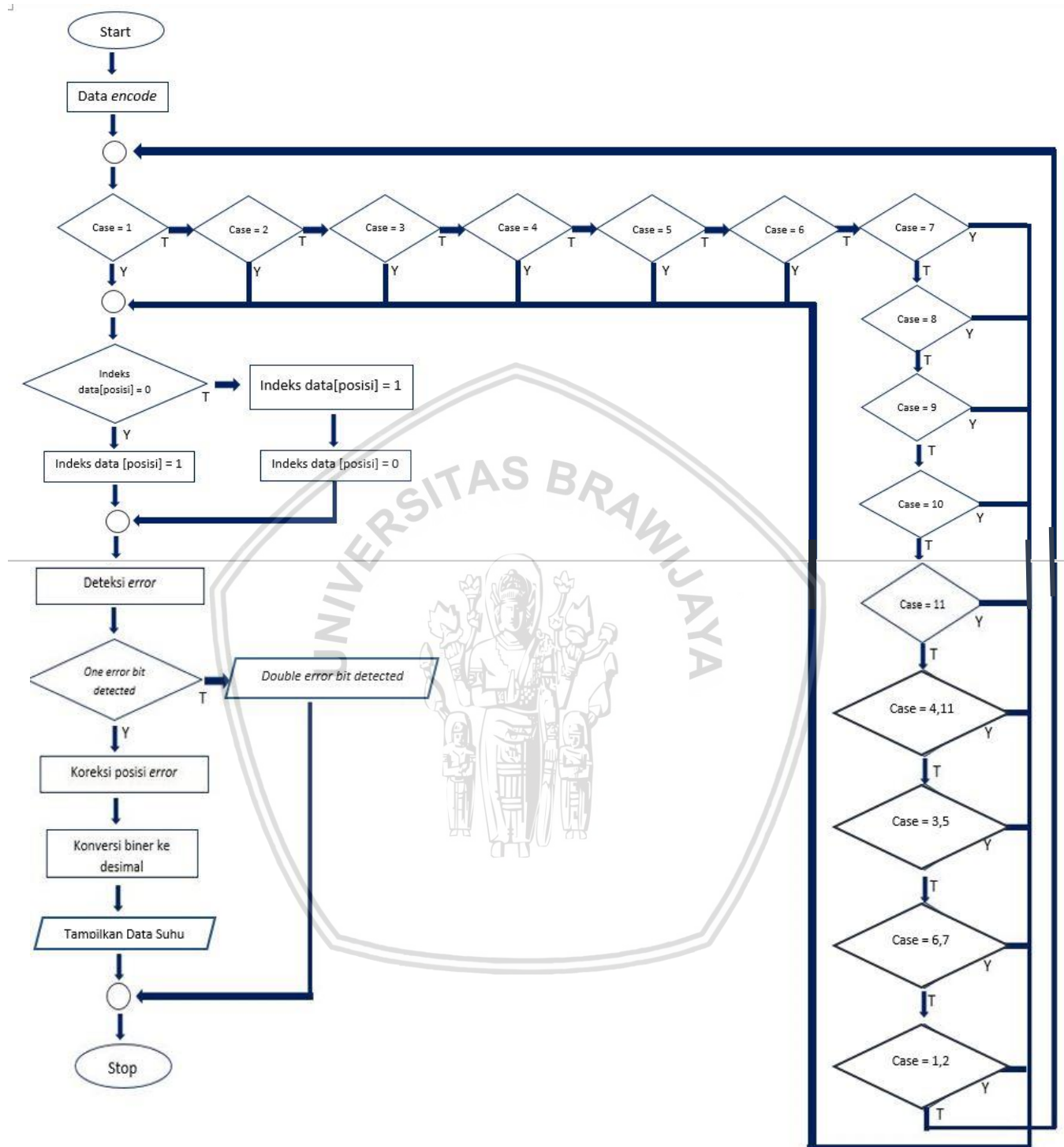
Posisi *parity bit* terakhir pada data *encode* yaitu  $P_x$ , akan dilakukan proses perhitungan dengan logika *XOR* jika hasil dari proses cek *error* bernilai lebih dari 0 yang berarti terdapat *error* data pada posisi tersebut. Namun, hasil dari proses cek *error* masih belum cukup memberikan informasi apakah *error* yang terjadi pada data *encode* berjumlah 1 bit atau lebih dari 1 bit.

Sistem pengujian dapat mendeteksi dua jenis *error*, yaitu 1 *bit error* dan 2 *bit error*. Letak posisi *error* akan dimasukkan berupa data *input* yang berupa tampilan menu. Pada pilihan 1-11, akan mengubah posisi antara posisi 1-11. Data yang terdapat pada posisi ini akan dirubah nilainya dalam bilangan biner. Kemudian terdapat pilihan (4,11), (3,5), (6,7), (9,10), dan (1,2) yang berarti akan mengubah data pada dua posisi. Metode *Hamming Code* yang diterapkan akan mampu mendeteksi jumlah *error* data yang dimasukkan pada menu pengujian.

Proses pada sistem pengujian, data yang dimasukkan melalui menu pengujian akan dimasukkan ke dalam kondisi *switch case*. Nilai dari *switch case* akan dimasukkan ke dalam variabel posisi yang akan dikurangi 1. Hal ini dilakukan karena dalam metode *Hamming Code* posisi data dimulai dari indeks 1 dan data *encode* posisi data dimulai dari indeks 0. Sehingga indeks ke 0 data *encode* sama dengan indeks ke 1 untuk deteksi dan koreksi *error Hamming Code*. Ketika posisi tersebut telah terisi nilainya, maka nilai dari posisi tersebut akan diubah dan setelah melewati proses tersebut data *encode* yang didapatkan akan mengalami kesalahan atau *error* sebanyak 1 bit atau 2 bit.

Nilai data pada pengujian diubah, data dimasukkan ke dalam proses deteksi *error*. Jika data pengujian terdeteksi mengalami *error* sebanyak 2 *bit*, maka sistem pengujian akan menampilkan deteksi *error* namun tidak dapat dilakukan koreksi data dikarenakan sifat dari metode *Hamming Code*. Ketika data pengujian mengalami *error* sebanyak 1 *bit*, maka sistem pengujian akan menampilkan deteksi *error* sekaligus melakukan koreksi pada *bit* tersebut. Selanjutnya data yang mengalami *error* sebanyak 1 *bit* akan dikonversi ke desimal dan ditampilkan dengan data yang sudah dikoreksi atau dibenarkan.

*Flowchart* sistem pengujian *decode* data dengan menggunakan metode *Hamming Code* yang digambarkan pada Gambar 5.12:



Gambar 5.12 Flowchart sistem pengujian data

Berdasarkan Gambar 5.12, Tabel 5.14 akan menunjukkan perancangan perhitungan deteksi *error* dengan data yang salah atau *error* :

**Tabel 5.14 Proses deteksi 1 bit error pada 12 bit data**

Posisi Data	P1	P2	D3	P4	D5	D6	D7	P8	D9	D10	D11	Px
Data yang dikirim	1	1	1	1	0	1	0	1	0	1	0	1
Data yang diterima	1	1	1	1	0	1	1	1	0	1	0	1
P1	$P1 \oplus D3 \oplus D5 \oplus D7 \oplus D9 \oplus D11 = 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 0 = 1$											
P2	$P2 \oplus D3 \oplus D6 \oplus D7 \oplus D10 \oplus D11 = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 1$											
P4	$P4 \oplus D5 \oplus D6 \oplus D7 = 1 \oplus 0 \oplus 1 \oplus 1 = 1$											
P8	$P8 \oplus D9 \oplus D10 \oplus D11 = 1 \oplus 0 \oplus 1 \oplus 0 = 0$											
Cek error	$(1 \times P1) + (2 \times P2) + (4 \times P4) + (8 \times P8) = (1 \times 1) + (2 \times 1) + (4 \times 1) + (8 \times 0) = 7$											
Px	$P1 \oplus P2 \oplus D2 \oplus D3 \oplus D4 \oplus D5 \oplus D6 \oplus D7 \oplus P8 \oplus D9 \oplus D10 \oplus D11 \oplus Px = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 = 1$											
Data encode	111101110101 = <i>error 1 bit</i> pada posisi 7											

Berdasarkan Tabel 5.14 pengujian menggunakan data 12 bit untuk dilakukan proses deteksi dan koreksi *error* dengan menghitung ulang data yang *encode* yang didapatkan dengan rumus metode *Hamming Code*. Ketika nilai hasil proses cek *error* bernilai lebih dari 0, maka menandakan adanya *error* pada data tersebut. Namun, masih belum bisa diketahui berapa jumlah *error* yang terjadi pada *bit* data *encode*.

Pada penelitian ini, membuktikan bahwa metode *Hamming Code* dapat mendeteksi lebih dari 1 *bit error* yaitu berjumlah 2 *bit error*. Untuk lebih memastikan jumlah *error* yang terjadi, maka ditambahkan *parity bit* tambahan yang terletak diakhir data. *Parity bit* ini tidak diikuti dalam proses operasi logika *XOR* pada tiap *parity* tambahan dan hanya dihitung dalam operasi logika *XOR* ketika nilai cek *error* bernilai lebih dari 0. Ketika nilai Px bernilai 1 maka akan terdeteksi 1 *bit* data yang *error*. Jika bernilai 0, maka terdeteksi 2 *bit* data yang *error*, namun tidak dapat mendeteksi pada posisi berapa letak *error* tersebut. Pada Tabel 5.15 akan menunjukkan proses perhitungan deteksi 2 *bit error* pada 12 bit data yang diterima.

Tabel 5.15 Proses deteksi 2 bit error pada 12 bit data

Posisi Data	P1	P2	D3	P4	D5	D6	D7	P8	D9	D10	D11	Px
Data yang dikirim	1	1	1	1	0	1	0	1	0	1	0	1
Data yang diterima	1	1	1	0	0	1	1	1	0	1	0	1
P1	$P1 \oplus D3 \oplus D5 \oplus D7 \oplus D9 \oplus D11 = 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 0 = 1$											
P2	$P2 \oplus D3 \oplus D6 \oplus D7 \oplus D10 \oplus D11 = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 1$											
P4	$P4 \oplus D5 \oplus D6 \oplus D7 = 0 \oplus 0 \oplus 1 \oplus 1 = 0$											
P8	$P8 \oplus D9 \oplus D10 \oplus D11 = 1 \oplus 0 \oplus 1 \oplus 0 = 0$											
Cek error	$(1 \times P1) + (2 \times P2) + (4 \times P4) + (8 \times P8) = (1 \times 1) + (2 \times 1) + (4 \times 0) + (8 \times 0) = 3$											
Px	$P1 \oplus P2 \oplus D2 \oplus D3 \oplus D4 \oplus D5 \oplus D6 \oplus D7 \oplus P8 \oplus D9 \oplus D10 \oplus D11 \oplus Px = 1 \oplus 1 \oplus 1 \oplus 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 = 0$											
Data encode	111001110101 = error 2 bit detected											

## 5.2 Implementasi

Pada subbab ini, akan menjelaskan secara rinci tentang implementasi metode *Hamming Code* hasil perancangan. Implementasi dilakukan ketika perancangan sistem sudah terpenuhi. Penjelasan setiap proses implementasi yang dilakukan akan dijelaskan pada subbab-subbab selanjutnya.

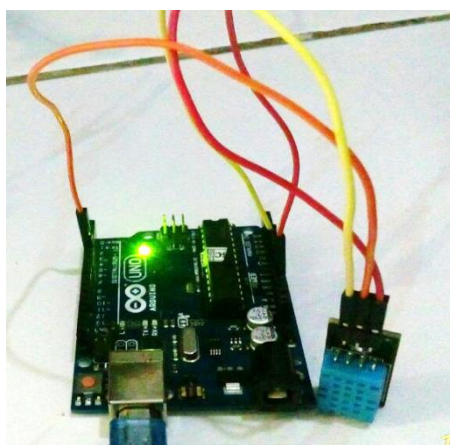
### 5.2.1 Implementasi Perangkat Keras

Setelah melalui tahap perancangan perangkat keras, selanjutnya adalah tahapan implementasi beberapa konfigurasi atau *library* yang dibutuhkan. Berikut adalah penjelasan lebih rinci tentang bagian-bagian implementasi perangkat keras.

#### 5.2.1.1 Implementasi Sensor DHT11 Pada Arduino Uno

Berdasarkan desain perancangan yang dilakukan, sensor DHT11 dihubungkan pada *Arduino Uno* untuk mendapatkan data suhu dan kelembaban. Proses implementasi ini juga sesuai dengan spesifikasi pin yang digunakan berdasarkan Tabel 5.1. Pada Gambar 5.13 akan menunjukkan implementasi sensor DHT11 pada *Arduino Uno*:





**Gambar 5.13 Implementasi sensor DHT11 pada Arduino Uno**

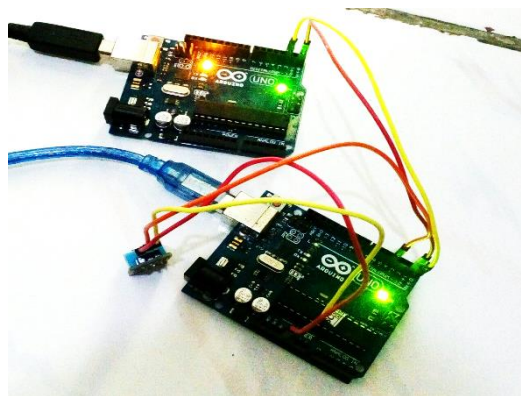
Berdasarkan Gambar 5.13, sensor DHT11 yang terhubung pada *Arduino Uno* dalam penggunaannya di program *Arduino IDE* yaitu dengan menggunakan *library* DHT11 pada awal program ditulis. Dengan menggunakan *library* ini, maka dapat memakai fungsi dari sensor DHT11 yaitu untuk mendapatkan data suhu. Pada Tabel 5.16 akan menunjukkan kode program untuk menggunakan fungsi *library* DHT11 sekaligus dengan mendeklarasikan variabel global untuk menyimpan data yang masuk dari sensor DHT11 :

**Tabel 5.16 Implementasi *library* sensor DHT11 pada Arduino Uno**

Kode Program	
1	<code>int x,bits[12],data[12];</code>
2	<code>#include &lt;DHT11.h&gt;</code>
3	<code>#include "DHT.h"</code>
4	<code>#define pinDHT 4</code>
5	<code>#define tipeDHT DHT11</code>
6	<code>DHT sensorDHT(pinDHT, tipeDHT);</code>

#### 5.2.1.2 Implementasi Antar Mikrokontroler Arduino Uno

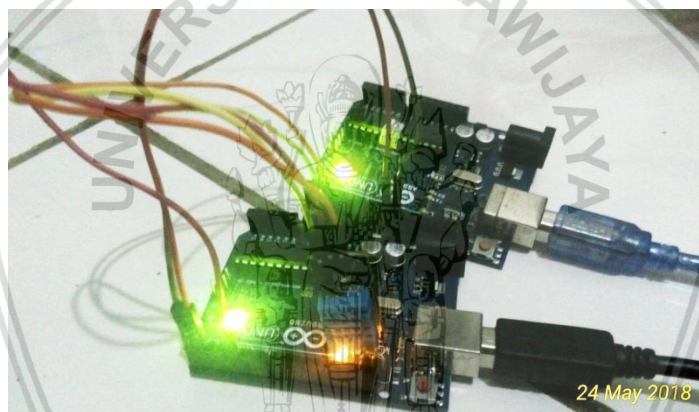
Berdasarkan skematik perancangan yang telah dijelaskan pada subbab perancangan, kedua mikrokontroler *Arduino Uno* yang dihubungkan sebagai sistem antar pengirim dan penerima data. Dimana pengirim data dihubungkan dengan sensor DHT11 untuk mendapatkan data suhu. Pada Gambar 5.14 akan menunjukkan implementasi antar *Arduino Uno*.



**Gambar 5.14 Implementasi antar mikrokontroller *Arduino Uno***

### **5.2.1.3 Implementasi Sistem Pengujian**

Berdasarkan pada perancangan skematik pengujian, implementasi antar *Arduino Uno* yang dirancang sebagai sistem pengujian dengan menggunakan dua pin berbeda sebagai *TX* dan *RX* pada bagian penerima. Pada Gambar 5.15 akan menunjukkan implementasi tersebut.



**Gambar 5.15 Implementasi sistem pengujian**

### **Implementasi Perangkat Lunak**

Subbab ini menjelaskan tentang implementasi pada perancangan perangkat lunak yang berupa *flowchart* metode *Hamming Code* menjadi kode program untuk perangkat keras.

#### **5.2.2.1 Implementasi Antar Muka Serial Pada Pengirim dan Penerima Data**

*Baudrate* yang digunakan antar pengirim dan penerima data menggunakan nilai *baudrate* 9600 yaitu nilai yang paling minimal dari program *Arduino IDE*. Pada Tabel 5.17 akan menunjukkan implementasi antar muka serial.

**Tabel 5.17 Antar muka serial pada pengirim dan penerima data *Arduino Uno***

Kode Program	
1	void setup() {
2	Serial.begin(9600);
3	}

**Tabel 5.18 Antar muka serial pada penerima data**

Kode Program	
1	void setup() {
2	Serial.begin(9600);
3	myserial.begin(9600);
	}

Pada Tabel 5.18, pada bagian penerima data, terdapat dua jenis masukan yaitu data suhu dan data pengujian yang berupa data masukan dari *PC*. Pada table tersebut terdapat dua *serial* yang memiliki fungsi berbeda. *Serial.begin(9600)* untuk menerima data antar mikrokontroller, sedangkan *myserial.begin(9600)* untuk menerima data antar mikrokontroller dan *PC*.

### 5.2.2.2 Implementasi Metode Hamming Code pada encode data

Sebelum melakukan proses *encode* data dengan menggunakan metode *Hamming Code*, data suhu terlebih dahulu dikonversikan ke biner. Tujuannya agar data tersebut dapat diolah dengan menggunakan metode *Hamming Code*.

Terdapat 4–5 *parity bit* tambahan yang disisipkan pada data sebelum dikirim. Jumlah *parity bit* ini tergantung dari angka desimal yang didapatkan dari data sensor yang kemudian dikonversikan ke biner. Pada penelitian ini, jumlah data biner hasil konversi sebanyak 3 bit – 7 bit data. Sehingga banyaknya bit dalam proses konversi desimal ke biner inilah yang akan menentukan jumlah *parity bit* yang ditambahkan pada data encode. Pada Tabel 5.19 akan menunjukkan kode dalam mengubah data *integer* menjadi biner yang ditunjukkan pada beberapa tabel dibawah ini :

**Tabel 5.19 Kode konversi data *integer* menjadi biner 3 bit**

Kode Program	
1	if(suhu<=7) {
2	x = 7;
3	Serial.write(x);
4	for(int b=0;b<=2;b++) {
5	bits[b] = suhu%2;
6	suhu = suhu/2;
7	}

Pada Tabel 5.19, terdapat kondisi untuk rentang bilangan desimal 0-7 akan dikonversikan menjadi biner sebanyak 3 bit. Dengan demikian data suhu dikonversikan menjadi 3 bit data yang akan dikirimkan dengan ditambahkan 4 *parity bit* tambahan pada proses *encoding*nya. Total data yang akan dikirimkan adalah 7 bit data, sehingga *id* yang dikirimkan ke penerima yaitu 7 sebagai panjang *array* untuk menampung data *encode*.

**Tabel 5.20 Kode konversi data *integer* menjadi biner 4 bit**

Kode Program	
1	else if(suhu>=8 && suhu<=15){
2	x = 8;
3	Serial.write(x);
4	for(int i=0;i<=3;i++){
5	bits[i] = suhu%2;
6	suhu = suhu/2;
7	}

Pada Tabel 5.20, kondisi rentang bilangan desimal 8-15 akan dikonversikan menjadi biner sebanyak 4 bit. Dengan demikian data suhu dikonversikan menjadi 4 bit data yang akan dikirimkan dengan ditambahkan 4 *parity bit* tambahan pada proses *encoding*-nya. Total data yang akan dikirimkan adalah 8 bit data, sehingga *id* yang dikirimkan ke penerima yaitu 8 sebagai panjang *array* untuk menampung data *encode*.

**Tabel 5.21 Kode konversi data *integer* menjadi biner 5 bit**

Kode Program	
1	else if(suhu>=16 && suhu<=31){
2	x = 10;
3	Serial.write(x);
4	for(int b=0;b<=4;b++){
5	bits[b] = suhu%2;
6	suhu = suhu/2;
7	}

Pada Tabel 5.21, kondisi rentang bilangan desimal 16-31 akan dikonversikan menjadi biner sebanyak 5 bit. Dengan demikian data suhu dikonversikan menjadi 5 bit data yang akan dikirimkan dengan ditambahkan 5 *parity bit* tambahan pada proses *encoding*-nya. Total data yang akan dikirimkan adalah 10 bit data, sehingga *id* yang dikirimkan ke penerima yaitu 10 sebagai panjang *array* untuk menampung data *encode*.

**Tabel 5.22 Kode konversi data *integer* menjadi biner 6 bit**

Kode Program	
1	else if(suhu>=32 && suhu<=63) {
2	x = 11;
3	Serial.write(x);
4	for(int b=0;b<=5;b++){
5	bits[b] = suhu%2;
6	suhu = suhu/2;
7	}

Pada Tabel 5.22, kondisi rentang bilangan desimal 32-63 akan dikonversikan menjadi biner sebanyak 6 bit. Dengan demikian data suhu dikonversikan menjadi 6 bit data yang akan dikirimkan dengan ditambahkan 5 *parity bit* tambahan pada proses *encoding*-nya. Total data yang akan dikirimkan adalah 11 bit data, sehingga *id* yang dikirimkan ke penerima yaitu 11 sebagai panjang *array* untuk menampung data *encode*.

**Tabel 5.23 Kode konversi data *integer* menjadi biner 7 bit**

Kode	Program
1	else if(suhu>=64 && suhu<=127){
2	x = 12;
3	Serial.write(x);
4	for(int b=0;b<6;b++){
5	bits[b] = suhu%2;
6	suhu = suhu/2;
7	}

Pada Tabel 5.23, kondisi rentang bilangan desimal 64-127 akan dikonversikan menjadi biner sebanyak 7 bit. Dengan demikian data suhu dikonversikan menjadi 7 bit data yang akan dikirimkan dengan ditambahkan 5 *parity bit* tambahan pada proses *encoding*-nya. Total data yang akan dikirimkan adalah 12 bit data, sehingga *id* yang dikirimkan ke penerima yaitu 12 sebagai panjang *array* untuk menampung data *encode*.

Setelah dilakukan proses konversi desimal ke biner pada data suhu, selanjutnya adalah mengimplementasikan metode *Hamming Code* pada bagian pengirim data. Data yang dihasilkan pada proses ini disebut data *encode*. Tabel 5.24 dan Tabel 5.25 akan menunjukkan *code* implementasi metode *Hamming Code* pada setiap jumlah bit data:

**Tabel 5.24 Implementasi metode *Hamming Code* pada *encode data*(3,7)**

Kode	Program
1	data[2] = bits[2];
2	data[5] = bits[0];
3	data[4] = bits[1];
4	data[0] = data[2] ^ data[4];
5	data[1] = data[2] ^ data[4] ^ data[5];
6	data[3] = data[4] ^ data[5];
7	data[6] = data[0] ^ data[1] ^ data[2] ^ data[3] ^ data[4] ^ data[5];
8	for(int i=0;i<7;i++){
9	Serial.write(data[i]);
10	}
11	}

**Tabel 5.25 Implementasi metode *Hamming Code* pada *encode data*(4,8)**

Kode	Program
1	data[2] = bits[3];
2	data[4] = bits[2];
3	data[5] = bits[1];
4	data[6] = bits[0];
5	data[0] = data[2] ^ data[4] ^ data[6];
6	data[1] = data[2] ^ data[5] ^ data[6];
7	data[3] = data[4] ^ data[5] ^ data[6];
8	data[7] = data[0] ^ data[1] ^ data[2] ^ data[3] ^ data[4] ^ data[5] ^ data[6];
9	for(int i=0;i<8;i++){
10	Serial.write(data[i]);
11	}
12	}

Berdasarkan pada Tabel 5.24 dan Tabel 5.25, sesuai dengan *flowchart* perancangan yang dituliskan pada subbab perancangan, data hasil konversi desimal ke biner ini dimasukkan pada variabel *array encode* sebelum data



dilakukan proses *encode*. Kemudian pada indeks *parity bit* tambahan yaitu 0,1,3,dan Px yang diisi dengan hasil operasi logika XOR antar data konversi. *Parity bit* Px ini terletak pada posisi terakhir data *encode* Kemudian data dikirim dengan dalam tipe *array* dengan menggunakan *serial.write* sesuai dengan jumlah data *encode*.

**Tabel 5.26 Implementasi metode *Hamming Code* pada *encode data(5,10)***

Kode Program	
1	data[2] = bits[4];
2	data[4] = bits[3];
3	data[5] = bits[2];
4	data[6] = bits[1];
5	data[8] = bits[0];
6	data[0] = data[2] ^ data[4] ^ data[6] ^ data[8];
7	data[1] = data[2] ^ data[5] ^ data[6];
8	data[3] = data[4] ^ data[5] ^ data[6];
9	data[7] = data[8];
10	data[9] = data[0]^data[1]^data[2]^data[3]^
11	data[4]^data[5]^data[6]
12	^data[7]^data[8];
13	for(int i=0;i<10;i++){
14	Serial.write(data[i]);
15	}
16	}

**Tabel 5.27 Implementasi metode *Hamming Code* pada *encode data(6,11)***

Kode Program	
1	data[2] = bits[5];
2	data[4] = bits[4];
3	data[5] = bits[3];
4	data[6] = bits[2];
5	data[8] = bits[1];
6	data[9] = bits[0];
7	data[0] = data[2] ^ data[4] ^ data[6] ^ data[8];
8	data[1] = data[2] ^ data[5] ^ data[6] ^ data[9];
9	data[3] = data[4] ^ data[5] ^ data[6];
10	data[7] = data[8] ^ data[9];
11	data[10]= data[0]^data[1]^data[2]^data[3]^data[4]
12	^data[5]^data[6]^data[7]^data[8]^data[9];
13	for(int i=0;i<11;i++){
14	Serial.write(data[i]);
15	}
16	}

**Tabel 5.28 Implementasi metode *Hamming Code* pada *encode data(7,12)***

Kode Program	
1	data[2] = bits[6];
2	data[4] = bits[5];
3	data[5] = bits[4];
4	data[6] = bits[3];
5	data[8] = bits[2];
6	data[9] = bits[1];
7	data[10] = bits[0];
8	data[0] = data[2] ^ data[4] ^ data[6] ^ data[8] ^ data[10];
9	data[1] = data[2] ^ data[5] ^ data[6] ^ data[9] ^ data[10];
10	data[3] = data[4] ^ data[5] ^ data[6];

11	<code>data[7] = data[8]^ data[9]^ data[10];</code>
12	<code>data[11] = data[0]^data[1]^data[2]^data[3]^</code>
13	<code>data[4]^data[5]^data[6]^data[7]^data[8]^</code>
14	<code>data[9]^data[10];</code>
15	<code>for(int i=0;i&lt;12;i++){</code>
16	<code>Serial.write(data[i]);</code>
17	<code>}</code>
18	<code>}</code>

Pada Tabel 5.26 – Tabel 5.28 , posisi *parity bit* yang ditambahkan yaitu pada posisi 0,1,3,7, dan Px. Kode pada Tabel 5.28 juga akan diterapkan pada sistem pengujian bagian pengirim.

### 5.2.2.3 Implementasi Metode Hamming Code Pada Decode Data

Berdasarkan *flowchart* perancangan metode *Hamming Code* pada *decode* data, sebelum data dimasukkan pada proses *decode*, data *encode* terlebih dahulu ditampung pada sebuah variabel *array* global yang ditunjukkan pada Tabel 5.29 :

**Tabel 5.29 Variabel penampung data *encode***

	Kode Program
1	<code>int dt[15],id,c1,c2,c4,c8,c,x,cx;</code>
2	<code>int plus,bin[15],tmp,hasil=0,konversi=0;</code>

Tujuan dari deklarasi beberapa variabel secara global agar data yang ditampung pada variabel-variabel tersebut dapat digunakan pada lain fungsi selain fungsi utama yaitu *void loop()*.

Pada Tabel 5.29 juga terdapat beberapa variabel bertipe *integer* dan *array* untuk menampung data *encode*. Selanjutnya pada Tabel 5.30 menunjukkan nilai dari variabel *id* yang akan menentukan panjang *array* untuk menampung data:

**Tabel 5.30 Variabel yang menampung data *encode***

	Kode Program
1	<code>void loop() {</code>
2	<code>id = Serial.read();</code>
3	<code>for(int i=0;i&lt;id;i++){</code>
4	<code>dt[i] = Serial.read();</code>
5	<code>Serial.print(dt[i]);</code>
6	<code>}</code>

Implementasi pada penelitian ini dibedakan menjadi 2 jenis sistem. Berikut adalah penjelasan lebih detail implementasi metode *Hamming Code* pada 2 jenis sistem tersebut.

### 5.2.2.4 Implementasi Metode Hamming Code Pada Penerima Data

Pada penerima data yaitu *Arduino Uno*, implementasi dilakukan sama persis dengan bagian penerima data, yaitu sistem dapat melakukan *decode* pada setiap jenis jumlah data *encode* yang dikirimkan. Tabel 5.31 dan Tabel 5.32 akan menunjukkan implementasi metode *Hamming Code* pada bagian penerima data:

**Tabel 5.31 Implementasi metode *Hamming Code* pada data *encode* 7 bit**

Kode Program	
1	if(id==7) {
2	c1=dt[2]^dt[4]^dt[0];
3	c2=dt[2]^dt[4]^dt[5];
4	c4=dt[3]^dt[4]^dt[5];
5	cx=dt[6]^dt[0]^dt[1]^dt[2]^dt[3]^dt[4]^dt[5];
6	c=c1+c2*2;
7	koreksi(c);

**Tabel 5.32 Implementasi metode *Hamming Code* pada data *encode* 8 bit**

Kode Program	
1	else if(id==8) {
2	c1=dt[2]^dt[4]^dt[6]^dt[0];
3	c2=dt[2]^dt[5]^dt[6]^dt[1];
4	c4=dt[4]^dt[5]^dt[6]^dt[3];
5	cx=dt[7]^dt[0]^dt[1]^dt[2]^dt[3]^dt[4]^dt[5]^
6	dt[6];
7	c=c1+c2*2+c4*4;
8	koreksi(c);
9	bin[0] = dt[6];
10	bin[1] = dt[5];
11	bin[2] = dt[4];
12	bin[3] = dt[2];
13	x = 3;
14	biner(x);
15	}

Berdasarkan Tabel 5.31 dan Tabel 5.32, data *encode* akan dimasukkan pada sebuah kondisi. Kondisi inilah yang membuat sistem dapat melakukan proses deteksi dan koreksi *error* dengan metode *Hamming Code* pada data 6-11 bit. Nilai variabel *id* akan menentukan proses cek *error* data karena nilai variabel ini merupakan panjang data *encode* yang diterima. Terdapat variabel *c1*, *c2*, *c4*, *c8*, dan *cx* yang merupakan variabel *parity bit* tambahan. Nilai dari variabel ini akan dihasilkan dari operasi logika XOR dari beberapa nilai biner data *encode*. Kemudian hasil nilai dari setiap variabel ini akan dikalikan dengan angka indeks *parity bit* tambahan dalam metode *Hamming Code* yaitu 1,2,4. Ketiga angka ini dapat merepresentasikan posisi data yang sesuai dengan jumlah datanya. Hasil dari perkalian tiap variabel dengan indeks *parity bit* tambahan akan ditambahkan dan ditampung dalam variabel *c* yang merupakan variabel penampung posisi *error* data. Variabel ini yang akan diletakkan pada sebuah fungsi koreksi *error*. Sedangkan untuk variabel *cx* dilakukan operasi logika XOR pada semua *bit* data *encode* yang diterima. Hasil dari variabel *cx* yang akan menentukan jumlah *error* yang terjadi pada data *encode*.

**Tabel 5.33 Implementasi metode *Hamming Code* pada data encode 10 bit**

Kode Program	
1	c1=dt[2]^dt[4]^dt[6]^dt[8]^dt[0];
2	c2=dt[2]^dt[5]^dt[6]^dt[1];
3	c4=dt[4]^dt[5]^dt[6]^dt[3];
4	c8=dt[7]^dt[8];
5	cx=dt[9]^dt[0]^dt[1]^dt[2]^dt[3]^dt[4]^
6	dt[5]^dt[6]^dt[7]^dt[8];
7	c=c1+c2*2+c4*4+c8*8;
8	koreksi(c);
9	bin[0] = dt[8];
10	bin[1] = dt[6];
11	bin[2] = dt[5];
12	bin[3] = dt[4];
13	bin[4] = dt[2];
14	x = 4;
15	biner(x);
16	}

**Tabel 5.34 Implementasi metode *Hamming Code* pada data encode 11 bit**

Kode Program	
1	c1=dt[2]^dt[4]^dt[6]^dt[8]^dt[0];
2	c2=dt[2]^dt[5]^dt[6]^dt[1];
3	c4=dt[4]^dt[5]^dt[6]^dt[3];
4	c8=dt[8]^dt[9]^dt[7];
5	cx=dt[10]^dt[0]^dt[1]^dt[2]^dt[3]^dt[4]^
6	dt[5]^dt[6]^dt[7]^dt[8]^dt[9];
7	c=c1+c2*2+c4*4+c8*8;
8	koreksi(c);
9	bin[0] = dt[9];
10	bin[1] = dt[8];
11	bin[2] = dt[6];
12	bin[3] = dt[5];
13	bin[4] = dt[4];
14	bin[5] = dt[2];
15	x = 5;
16	biner(x);
17	}

**Tabel 5.35 Implementasi metode *Hamming Code* pada data encode 12 bit**

Kode Program	
1	c1=dt[2]^dt[4]^dt[6]^dt[8]^dt[10]^dt[0];
2	c2=dt[2]^dt[5]^dt[6]^dt[9]^dt[10]^dt[1];
3	c4=dt[4]^dt[5]^dt[6]^dt[3];
4	c8=dt[8]^dt[9]^dt[10]^dt[7];
5	cx=dt[11]^dt[0]^dt[1]^dt[2]^dt[3]^dt[4]^
6	dt[5]^dt[6]^dt[7]^dt[8]^dt[9]^dt[10];
7	c=c1+c2*2+c4*4+c8*8;
8	koreksi(c);
9	bin[0] = dt[10];
10	bin[1] = dt[9];
11	bin[2] = dt[8];
12	bin[3] = dt[6];
13	bin[4] = dt[5];
14	bin[5] = dt[4];
15	bin[6] = dt[2];
16	x = 6;
17	biner(x); }

Pada Tabel 5.33 – Tabel 5.35, kondisi pada panjang data 10,11 dan 12 bit menggunakan variabel c1,c2,c4,c8, dan cx yang berarti terdapat 5 *parity bit* tambahan yang terdapat dalam data *encode* tersebut.

Kemudian setelah data *encode* melalui proses cek *error* dengan menggunakan metode *Hamming Code*, nilai dari variabel c yang dimasukkan pada fungsi koreksi akan dilakukan proses koreksi data *encode* jika data tersebut mengalami *error*. Tabel 5.36 akan menunjukkan kode dari fungsi koreksi tersebut:

**Tabel 5.36 Fungsi koreksi data *encode***

Kode Program	
1	void koreksi(int data){
2	if(data==0){
3	Serial.println("No error");
4	}else{
5	if(cx == 0){
6	Serial.println("double error detected");
7	}else if(cx == 1){
8	Serial.println("one error detected..");
9	Serial.print("position: ");
10	Serial.println(data);
11	Serial.print("correct data: ");
12	int pos = data-1;
13	if(dt[pos] == 1){
14	dt[pos] = 0;
15	}else {
16	dt[pos] = 1;
17	}
18	}
19	}
20	}

Berdasarkan Tabel 5.36, jika nilai variabel c adalah 0, maka dalam rumus *Hamming Code* data tersebut bernilai benar dan tidak mengalami *error*. Namun jika data tersebut tidak bernilai 0, maka data tersebut mengalami *error*. Kemudian kondisi ini akan bercabang lagi jika nilai *parity* cx bernilai 0 maka terjadi *error* berjumlah 2 *bit* data. Sedangkan jika nilai *parity* cx bernilai 1 maka data mengalami *error* sebanyak 1 *bit* data.

Metode *Hamming Code* hanya mampu mengoreksi *error* sebanyak 1 *bit* data. Sehingga pada kondisi *error* 1 *bit* data maka posisi data *error* akan ditunjukkan oleh variabel pos, dimana nilai data dikurangi 1. Hal ini dilakukan karena indeks data *encode* dimulai dari indeks ke 0, sedangkan pada metode *Hamming Code* posisi data dimulai dari indeks ke 1. Selanjutnya data pada posisi tersebut akan dikoreksi dari 1 menjadi 0 dan sebaliknya. Proses koreksi pada metode *Hamming Code* hanya akan mengoreksi 1 bit data dengan nilai biner yaitu 0 atau 1.

Ketika data telah memasuki proses pada fungsi koreksi, proses selanjutnya adalah menampilkan data *encode* menjadi data suhu yang dikirimkan. Data akan ditampilkan dalam tipe *integer*. Data *encode* ini akan dimasukkan pada variabel *array* data biner yang akan ditunjukkan pada Tabel 5.37.



**Tabel 5.37 Data *encode* dimasukkan pada variabel data biner 3 bit**

Kode Program	
1	<code>bin[0] = dt[5];</code>
2	<code>bin[1] = dt[4];</code>
3	<code>bin[2] = dt[2];</code>
4	<code>x = 2;</code>
5	<code>biner(x);</code>
6	<code>}</code>

Pada Tabel 5.37 menunjukkan jika data suhu berjumlah 3 *bit* maka data hasil proses koreksi akan dimasukkan pada indeks *array* ke 0,1,dan 2. Serta memasukkan nilai x yang merepresentasikan nilai *array* untuk konversi biner ke desimal ke dalam fungsi *biner()*.

**Tabel 5.38 Data *encode* dimasukkan pada variabel data biner 4 bit**

Kode Program	
1	<code>bin[0] = dt[6];</code>
2	<code>bin[1] = dt[5];</code>
3	<code>bin[2] = dt[4];</code>
4	<code>bin[3] = dt[2];</code>
5	<code>x = 3;</code>
6	<code>biner(x);</code>
7	<code>}</code>

Tabel 5.38 menunjukkan jika data suhu berjumlah 4 *bit* maka data hasil proses koreksi akan dimasukkan pada indeks *array* ke 0,1,2,dan 3.

**Tabel 5.39 Data *encode* dimasukkan pada variabel data biner 5 bit**

No	Kode Program
1	<code>bin[0] = dt[8];</code>
2	<code>bin[1] = dt[6];</code>
3	<code>bin[2] = dt[5];</code>
4	<code>bin[3] = dt[4];</code>
5	<code>bin[4] = dt[2];</code>
6	<code>x = 4;</code>
7	<code>biner(x);</code>
8	<code>}</code>

Tabel 5.39 menunjukkan jika data suhu berjumlah 5 *bit* maka data hasil proses koreksi akan dimasukkan pada indeks *array* ke 0,1,2,3,dan 4.

**Tabel 5.40 Data *encode* dimasukkan pada variabel data biner 6 bit**

Kode Program	
1	<code>bin[0] = dt[9];</code>
2	<code>bin[1] = dt[8];</code>
3	<code>bin[2] = dt[6];</code>
4	<code>bin[3] = dt[5];</code>
5	<code>bin[4] = dt[4];</code>
6	<code>bin[5] = dt[2];</code>
7	<code>x = 5;</code>
9	<code>biner(x);</code>
10	<code>}</code>

Tabel 5.40 menunjukkan jika data suhu berjumlah 6 *bit* maka data hasil proses koreksi akan dimasukkan pada indeks *array* ke 0,1,2,3,4,dan 5.

**Tabel 5.41 Data *encode* dimasukkan pada variabel data biner 7 bit**

Kode Program	
1	<code>bin[0] = dt[10];</code>
2	<code>bin[1] = dt[9];</code>
3	<code>bin[2] = dt[8];</code>
4	<code>bin[3] = dt[6];</code>
5	<code>bin[4] = dt[5];</code>
6	<code>bin[5] = dt[4];</code>
7	<code>bin[6] = dt[2];</code>
9	<code>x = 6;</code>
10	<code>biner(x);</code>
11	<code>}</code>

Pada Tabel 5.41 menunjukkan jika data suhu berjumlah 7 *bit* maka data hasil proses koreksi akan dimasukkan pada indeks *array* ke 0,1,2, 3,4,5,dan 6.

Setelah data *encode* dimasukkan pada variabel *array* data biner, proses selanjutnya yaitu memasukkan data tersebut kedalam proses konversi biner ke desimal untuk mendapatkan data suhu yang dikirim. Terdapat nilai variabel *x* yang dimasukkan pada fungsi biner.

Variabel ini menyesuaikan dengan panjang *array* data yang sudah dimasukan dan yang menentukan panjang konversi biner ke desimal. Pada Tabel 5.42 akan menunjukkan kode konversi biner ke desimal:

**Tabel 5.42 Fungsi konversi biner ke desimal**

Kode Program	
1	<code>void biner(int x) {</code>
2	<code>for(int i=x;i&gt;=0;i--){</code>
3	<code>tmp = pow(2,i);</code>
4	<code>if(tmp &gt; 2){</code>
5	<code>tmp = tmp + 1;</code>
6	<code>}</code>
7	<code>hasil = bin[i]*tmp;</code>
9	<code>konversi = konversi+hasil;</code>
10	<code>}</code>
11	<code>Serial.print(konversi);</code>
12	<code>Serial.println("C");</code>
13	<code>konversi=0;</code>
14	<code>}</code>

Variabel *x* pada Tabel 5.42 sebagai parameter dari nilai variabel *x*. Nilai parameter ini akan menentukan nilai pangkat dari operasi biner ke desimal. Pada variabel *array bin[i]* merupakan variabel dari data *encode* yang telah dilakukan proses deteksi dan koreksi *error* menjadi data biner yang dikonversikan ke desimal dan ditampilkan menjadi data suhu yang dikirim.

### 5.2.2.5 Implemmentasi Metode Hamming Code Pada Sistem Pengujian

Sesuai dengan perancangan pada sistem pengujian,p engujian ini lebih difokuskan pada posisi *error* data. Sehingga pengujian ini akan menguji semua letak posisi data *encode* yang akan dibuat menjadi *error*. Pengujian ini juga menggunakan 2 serial yang akan menampung data *encode* dan data input menu

untuk uji coba posisi *error*. Pada Tabel 5.43 akan menunjukkan beberapa variabel dan serial yang diinisialisasi terlebih dahulu.

**Tabel 5.43 Inisialisasi variabel dan serial**

Kode Program	
1	#include <SoftwareSerial.h>
2	int rx[15],kx,plus,bin[10],binx[10],bink[10],no;
3	int c1,c2,c4,c8,c,cx;
4	int tmp,hasil=0,konversi=0;
5	int tmpx,hasilx=0,konversix=0;
6	SoftwareSerial myserial(10,11);
7	void setup() {
8	Serial.begin(9600);
9	myserial.begin(9600);
10	for (int i=0;i<=8;i++){
11	Serial.print("=");
12	delay(200);
13	}
14	Serial.println("");
15	Serial.println("MENU PENGUJIAN");
16	Serial.println("(a)Tampilkan Data Encode");
17	Serial.println("(b)Tampilkan Data Suhu");
18	Serial.println("Pengujian Error Bit Data ke : ");
19	Serial.println("(One bit error)1-10");
20	Serial.println("(Double bit error
21	detection)4,11/3,5/6,7/9,10/1,2");
22	Serial.println("Masukkan pilihan!");
23	}

Setelah memberi inisialisasi awal pada beberapa variabel global,serial, dan *library* kemudian menampung data *encode* dan data input dari serial monitor untuk melakukan pengujian. Kode ditunjukkan pada Tabel 5.44.

**Tabel 5.44 Variabel yang menampung data serial**

Kode Program	
1	void loop() {
2	if(Serial.available() > 0 ) {
3	int pil = Serial.read();

Variabel *pil* pada Tabel 5.44 merupakan variabel untuk menampung data yang didapatkan dari serial monitor. Sedangkan pada Tabel 5.45, terdapat beberapa menu untuk melakukan pengujian yang akan menggunakan *swicth case* dalam penerapannya. Pilihan *a* dan *b* digunakan untuk menampilkan data *encode* dalam biner dan dalam bentuk konversi desimal. Kode untuk pilihan *a* dan *b* yang ditunjukkan pada Tabel 5.45.

**Tabel 5.45 Kode menu pilihan *a* dan *b***

Kode Program	
1	switch (pil) {
2	case 'a' :
3	Serial.println("a");
4	for (int i=0;i<=11;i++){
5	rx[i] = myserial.read();
6	Serial.print(rx[i]);
7	}Serial.println("");

8	Serial.println("Masukkan pilihan!");
9	break;
10	case 'b' :
11	for (int i=0;i<11;i++){
12	rx[i] = Serial1.read();
13	}
14	bin[0] = rx[10];
15	bin[1] = rx[9];
16	bin[2] = rx[8];
17	bin[3] = rx[6];
18	bin[4] = rx[5];
19	bin[5] = rx[4];
20	bin[6] = rx[2];
21	for (int cx=6;cx>=0;cx--){
22	Serial.print(bin[cx]);
23	}
24	Serial.print(" = ");
25	for(int a=6;a>=0;a--){
26	tmp = pow(2,a);
27	hasil = bin[a]*tmp;
28	konversi = konversi+hasil;
29	}
30	Serial.println(konversi);
31	konversi=0;
32	Serial.println("Masukkan pilihan!");
33	break;

Pada Tabel 5.45, kita dapat melihat data *encode* sebelum dilakukan pengujian. Data ini akan dikonversi ke desimal dan ditampilkan tanpa dilakukan proses deteksi dan koreksi *error*. Selanjutnya adalah kode yang menunjukkan beberapa menu pilihan untuk melakukan pengujian letak posisi 1 *error* dan 2 *error* pada data *encode* yang ditunjukkan pada Tabel 5.46.

**Tabel 5.46 Kode menu pilihan untuk pengujian 1 bit error**

Kode Program	
1	case '1' :
2	Serial.println("1");
3	oneerror(1);
4	break;
5	case '2' :
6	Serial.println("2");
7	oneerror(2);
8	break;
9	case '3' :
10	Serial.println("3");
11	oneerror(3);
12	break;
13	case '4' :
14	Serial.println("4");
15	oneerror(4);
16	break;
17	case '5' :
18	Serial.println("5");
19	oneerror(5);
20	break;
21	case '6' :
22	Serial.println("6");
23	oneerror(6);
24	break;
25	case '7' :
26	Serial.println("7");
27	oneerror(7);
28	break;

29	case '8' :
30	Serial.println("8");
31	oneerror(8);
32	break;
33	case '9' :
34	Serial.println("9");
35	oneerror(9);
36	break;
37	case 'c' :
38	Serial.println("10");
39	oneerror(10);
40	break;
41	case 'v' :
42	Serial.println("11");
43	oneerror(11);
44	break;

**Tabel 5.47 Kode menu pilihan untuk pengujian 2 bit error**

No	Kode Program
1	case 'g' :
2	Serial.println("posisi 4 dan 11 error");
3	doublerror(4,11);
4	break;
5	case 'h' :
6	Serial.println("posisi 3 dan 5 error");
7	doublerror(3,5);
8	break;
9	case 'j' :
10	Serial.println("posisi 6 dan 7 error");
11	doublerror(6,7);
12	break;
13	case 'k' :
14	Serial.println("posisi 9 dan 10 error");
15	doublerror(9,10);
16	break;
17	case 'l' :
18	Serial.println("posisi 1 dan 2 error");
19	doublerror(1,2);
20	break;
21	default :
22	Serial.println("Masukkan sesuai menu pilihan!");
23	}

Pada Tabel 5.47 letak posisi *error* yang berjumlah 1 bit error akan dimasukkan pada fungsi *oneerror()*, sedangkan pada Tabel 5.44 letak posisi *error* yang berjumlah 2 bit error akan dimasukkan pada fungsi *doublerror()*. Tabel 5.48 dan Tabel 5.49 adalah tabel yang akan menunjukkan kode fungsi *oneerror()* dan *doublerror()*.

**Tabel 5.48 Fungsi *oneerror()* untuk pengujian 1 bit error**

Kode Program
1 void oneerror (int inp){
2 Serial.print("Data asli: ");
3 for (int bx=0;bx<12;bx++){
4 Serial.print(rx[bx]);
5 }
6 Serial.println("");
7 Serial.print("Data pengujian: ");
8 int index = inp - 1;
9 if(rx[index] == 1){



10	rx[index] = 0;
11	for (int by=0;by<=11;by++){
12	Serial.print(rx[by]);
13	} Serial.println("");
14	} else {
15	rx[index] = 1;
16	for (int by=0;by<11;by++){
17	Serial.print(rx[by]);
18	} Serial.println();
19	}
20	koreksi();
21	}

**Tabel 5.49 Fungsi *doublerror()* untuk pengujian 2 bit error**

Kode Program	
1	void doublerror (int a, int b){
2	int ax = a-1;
3	int bx = b-1;
4	if(rx[ax]== 0){
5	rx[ax]=1;
6	}else{
7	rx[ax]=0;
8	}
9	if(rx[bx]== 0){
10	rx[bx]=1;
11	}else{
12	rx[bx]=0;
13	}Serial.println("data pengujian:");
14	for (int i=0;i<11;i++){
15	Serial.print(rx[i]);
16	}Serial.println("");
17	koreksi();
18	}

Pada Tabel 5.48 , data yang terdapat dalam fungsi *onerror()* akan dijadikan sebuah parameter yaitu dengan variabel *inp* dalam tipe *integer*. Kemudian, karena indeks data *encode* dimulai pada posisi 0, maka nilai dari variabel *inp* ini akan dikurangi dengan 1 yang ditampung pada variabel *index*. Nilai dari variabel *index* ini akan ditujukan pada posisi data *encode* dan data *encode* pada posisi ini akan diubah.

Pola implementasi ini sama dengan kode pada Tabel 5.49, hanya saja pada fungsi *doublerror()* terdapat dua data yang menjadi parameter variabel posisi data yaitu variabel *a* dan *b*. Kemudian data yang dilewatkan pada kedua fungsi akan diubah nilainya dan akan menjadi data pengujian untuk cek deteksi dan koreksi *error* metode *Hamming Code*. Tabel 5.50 menunjukkan kode untuk deteksi dan koreksi *error*.

**Tabel 5.50 Deteksi error *Hamming Code* 12 bit data**

Kode Program	
1	c1=rx[2]^rx[4]^rx[6]^rx[8]^rx[10]^rx[0];
2	c2=rx[2]^rx[5]^rx[6]^rx[9]^rx[10]^rx[1];
3	c4=rx[4]^rx[5]^rx[6]^rx[3];
4	c8=rx[8]^rx[9]^rx[10]^rx[7];
5	cx=rx[11]^rx[0]^rx[1]^rx[2]^rx[3]^rx[4]^rx[5]^
6	rx[6]^rx[7]^rx[8]^rx[9]^rx[10];
7	c=c1+c2*2+c4*4+c8*8;

Berdasarkan Tabel 5.50, deteksi *error* menggunakan 12 bit data yang diterima. Terdapat 5 *parity bit* tambahan yang dilakukan proses cek *error* pada posisi 0,1,4,8,dan 12 yang direpresentasikan pada variabel c1,c2,c4,c8,dan cx. Hasil cek *error* ini akan ditampung pada variabel c yang menunjukkan posisi data *error* dan yang akan dilakukan proses koreksi *error*. Namun terdapat kondisi pada variabel cx yang menentukan jumlah *error* pada data *encode* tersebut. Tabel 5.51 di bawah ini menunjukkan kode untuk kondisi deteksi *error 1 bit* dan *2 bit* :

**Tabel 5.51 Kondisi deteksi *error 1 bit* dan *2 bit***

Kode Program	
1	if(c==0){
2	Serial.println("No error");
3	} else {
4	if(cx == 1){
5	Serial.print("one bit error detected.. ");
6	Serial.print("position: ");
7	Serial.println(c);
8	Serial.print("Error Correction: ");
9	int cx = c - 1;
10	if(rx[cx] == 0) {
11	rx[cx] = 1;
12	} else {
13	rx[cx] = 0;
14	}
15	for(int i=0;i<11;i++){
16	Serial.print(rx[i]);
17	}
18	Serial.println("");

Berdasarkan Tabel 5.51, ketika *error* berjumlah 2 *bit* maka hanya akan menampilkan *error 2 bit* yang terdeteksi namun tidak dapat mendeteksi diposisi berapa *error* itu terjadi. Berbeda dengan *error* yang berjumlah 1 *bit* yang akan mendeteksi sekaligus dapa melakukan koreksi data pada posisi *error* tersebut.

Tahapan terakhir dari sistem pengujian ini, ketika data telah dilakukan proses koreksi *error* maka data *encode* akan ditampilkan kembali dalam bentuk biner dan desimal. Tabel 5.52 akan menunjukkan kode untuk melakukan konversi biner ke desimal.

**Tabel 5.52 Konversi biner ke desimal**

Kode Program	
1	for(int i=0;i<11;i++){
2	Serial.print(rx[i]);
3	}
4	Serial.println("");
5	binx[0] = rx[10];
6	binx[1] = rx[9];
7	binx[2] = rx[8];
8	binx[3] = rx[6];
9	binx[4] = rx[5];
10	binx[5] = rx[4];
11	binx[6] = rx[2];
12	for (int ix=6;ix>=0;ix--){
13	Serial.print(binx[ix]);
14	}
15	Serial.print(" = ");
16	for(int i=6;i>=0;i--){
17	tmpx = pow(2,i);

18	hasilx = binx[i]*tmpx;
19	konversix = konversix+hasilx;
20	}
21	Serial.print(konversix);
22	Serial.println("C");
23	konversix=0;

Pada kode Tabel 5.52, baris perulangan *for* digunakan untuk menampilkan data *encode* dalam bentuk biner hasil proses koreksi. Kemudian data *encode* dimasukkan pada variabel *array* konversi biner ke desimal dan selanjutnya dilakukan proses konversi biner ke desimal serta ditampilkan melalui variabel *konversix*.



## BAB 6 PENGUJIAN DAN ANALISIS

Pengujian dan analisis dilakukan untuk mengetahui hasil penelitian yang dilakukan apakah sistem berjalan sesuai dengan kebutuhan fungsional dan non fungsional. Bab ini juga menjelaskan lebih detail tentang hasil pengujian yang dilakukan. Selain itu terdapat penjelasan tentang tujuan, prosedur, dan analisis yang dijabarkan pada setiap pengujian. Pengujian akan dibagi menjadi 4 jenis yaitu pengujian fungsional, non-fungsional, data *error*, dan waktu eksekusi.

### 6.1 Pengujian Fungsional

#### 6.1.1 Tujuan Pengujian

Pengujian dilakukan untuk mengetahui apakah sistem dapat melakukan kebutuhan fungsional yang dirancang pada bab rekayasa kebutuhan. Kebutuhan fungsional tersebut adalah sistem dapat membaca data suhu dan kelembaban.

#### 6.1.2 Prosedur Pengujian



**Gambar 6.1** Pengujian fungsional antar *arduino uno*

Berdasarkan Gambar 6.1, tahapan prosedur ini yang dilakukan untuk melakukan pengujian ini yaitu :

1. Menghubungkan sensor DHT11 pada mikrokontroller *Arduino Uno* yang berperan sebagai pengirim pada pin yang telah dituliskan pada bab perancangan yang akan mengambil data suhu.
2. Pin *UART Arduino Uno* pada pengirim dengan pin digital yang berperan sebagai *UART* pada *Arduino Uno* penerima dihubungkan menggunakan kabel *jumper*.
3. Menghubungkan kedua mikrokontroller *Arduino Uno* dengan laptop menggunakan kabel *usb*.
4. Meng-*upload* kode program pada bagian pengirim dan penerima.
5. Melihat hasil pembacaan sensor pada *Serial Monitor*.

### 6.1.3 Hasil dan Analisis Pengujian

```
(b) Pengujian ke 1
= 26 C
b
(b) Pengujian ke 2
= 26 C
b
(b) Pengujian ke 3
= 26 C
```

Gambar 6.2 Pengujian data suhu

Berdasarkan Gambar 6.1, sistem dapat membaca data sensor DHT11 yang bertipe *integer* antar mikrokontroller *Arduino Uno* yang dikirim sesuai dengan kebutuhan fungsional.

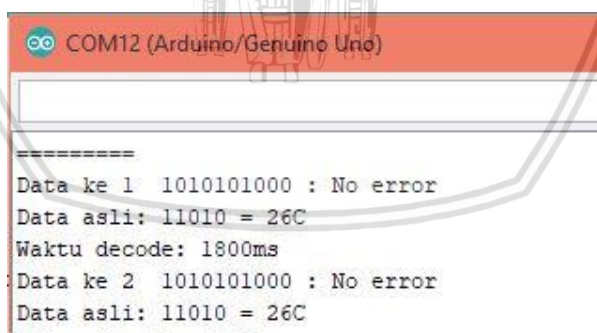
## 6.2 Pengujian Non Fungsional

### 6.2.1 Tujuan Pengujian

Tujuan dari pengujian ini yaitu untuk mengetahui apakah sistem dapat melakukan proses *encode* dan *decode data* dengan menggunakan metode *Hamming Code* serta melakukan proses konversi biner ke desimal. Hasil dari pengujian ini dilihat pada bagian penerima. Data *encode* akan ditampilkan pada bagian penerima dan akan dilakukan proses *decode* untuk menjalankan proses deteksi dan koreksi *error*.

### 6.2.2 Prosedur Pengujian

Pada pengujian nonfungsional, Tampilan untuk hasil pengujian yang telah dilakukan pada serial monitor ditampilkan pada Gambar 6.3.



```
COM12 (Arduino/Genuine Uno)

=====
Data ke 1  1010101000 : No error
Data asli: 11010 = 26C
Waktu decode: 1800ms
Data ke 2  1010101000 : No error
Data asli: 11010 = 26C
...
```

Gambar 6.3 Tampilan pengujian fungsional

Berdasarkan hasil pengujian pada Gambar 6.2 tahapan prosedur yang dilakukan untuk melakukan pengujian ini yaitu :

1. Menghubungkan sensor DHT11 pada mikrokontroller *Arduino Uno* yang berperan sebagai pengirim dan menghubungkan pin *UART* antar *Arduino Uno*.
2. Mengubah data suhu menjadi biner.
3. Melakukan proses *encode* dengan metode *Hamming Code* yang telah dirancang dan diimplementasikan dan kemudian dikirimkan.



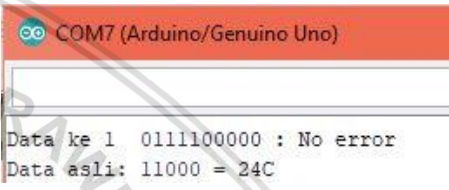
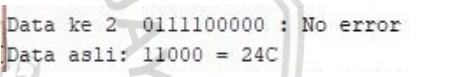
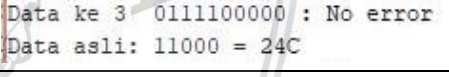
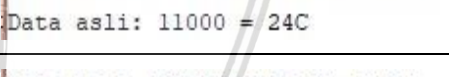
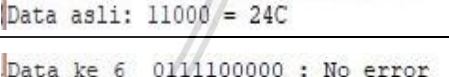
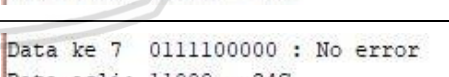
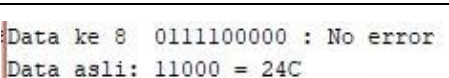
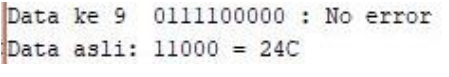
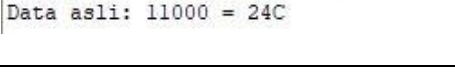
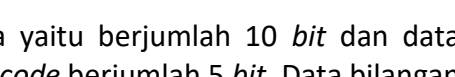
4. Data *encode* yang diterima dimasukkan pada proses deteksi dan koreksi *error* dengan metode *Hamming Code* sesuai dengan jumlah data yang diterima.

5. Hasil proses deteksi dan koreksi *error* diubah kembali menjadi desimal untuk menampilkan data suhu pada serial monitor.

### 6.2.3 Hasil dan Analisis Pengujian

Tabel 6.1 merupakan tabel yang menunjukkan hasil pengujian non fungsional yang dilakukan antar mikrokontroller *Arduino Uno* :

**Tabel 6.1 Hasil pengujian non fungsional pada suhu 24°C**

No	Nama Pengujian	Data Berhasil Terkirim	Hasil <i>decode</i>	Screenshoot
1	Pengujian ke-1	Ya	<i>No error</i>	
2	Pengujian ke-2	Ya	<i>No error</i>	
3	Pengujian ke-3	Ya	<i>No error</i>	
4	Pengujian ke-4	Ya	<i>No error</i>	
5	Pengujian ke-5	Ya	<i>No error</i>	
6	Pengujian ke-6	Ya	<i>No error</i>	
7	Pengujian ke-7	Ya	<i>No error</i>	
8	Pengujian ke-8	Ya	<i>No error</i>	
9	Pengujian ke-9	Ya	<i>No error</i>	
10	Pengujian ke-10	Ya	<i>No error</i>	

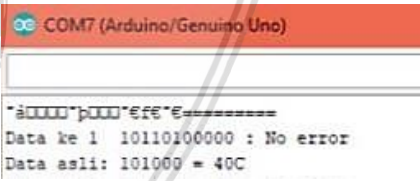
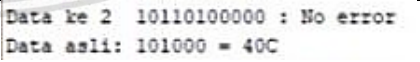

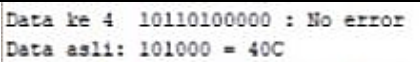
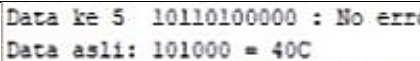
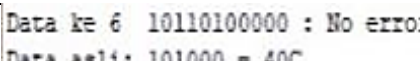
Berdasarkan Tabel 6.1, data yang terima yaitu berjumlah 10 *bit* dan data bilangan biner yang didapatkan dari proses *decode* berjumlah 5 *bit*. Data bilangan

biner tersebut berhasil dikonversi ke desimal menjadi data suhu asli tersebut yaitu 24°C. Perhitungan manual metode *Hamming Code* dengan menggunakan data yang sama seperti pada Tabel 6.1 yang ditunjukkan pada Tabel 6.2 untuk membuktikan bahwa hasil proses deteksi *error* tersebut benar.

**Tabel 6.2 Perhitungan manual deteksi *error* pada data suhu 24°C**

Data	P1	P2	D3	P4	D5	D6	D7	P8	D9	Px
0111100000	0	1	1	1	1	0	0	0	0	0
P1	$P1 \oplus D3 \oplus D5 \oplus D7 \oplus D9 = 0 \oplus 1 \oplus 1 \oplus 0 \oplus 0 = 0$									
P2	$P2 \oplus D3 \oplus D6 \oplus D7 = 1 \oplus 1 \oplus 0 \oplus 0 = 0$									
P4	$P4 \oplus D5 \oplus D6 \oplus D7 = 1 \oplus 1 \oplus 0 \oplus 0 = 0$									
P8	$D8 \oplus D9 = 0 \oplus 0 = 0$									
Cek Error	$(1 \times P1) + (2 \times P2) + (4 \times P4) + (8 \times P8) = (1 \times 0) + (2 \times 0) + (4 \times 0) + (8 \times 0) = 0$									
Data encode	0111100000 = no error									
Data suhu	D3,D5,D6,D7,D9 = 11000 = 24									

**Tabel 6.3 Hasil pengujian non fungsional pada suhu 40°C**

No	Nama Pengujian	Data Berhasil Terkirim	Hasil decode	Screenshoot
1	Pengujian ke-1	Ya	No error	
2	Pengujian ke-2	Ya	No error	
3	Pengujian ke-3	Ya	No error	
4	Pengujian ke-4	Ya	No error	
5	Pengujian ke-5	Ya	No error	
6	Pengujian ke-6	Ya	No error	

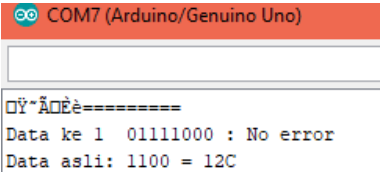
7	Pengujian ke-7	Ya	No error	Data ke 7 10110100000 : No error Data asli: 101000 = 40C
8	Pengujian ke-8	Ya	No error	Data ke 8 10110100000 : No error Data asli: 101000 = 40C
9	Pengujian ke-9	Ya	No error	Data ke 9 10110100000 : No error Data asli: 101000 = 40C
10	Pengujian ke-10	Ya	No error	Data ke 10 10110100000 : No error Data asli: 101000 = 40C

Pada Tabel 6.3 menggunakan data suhu 40°C yang dikirim dari sensor DHT11. Gambar pengujian yang berjumlah 10 pengujian menunjukkan bahwa data yang diterima tidak mengalami *error*. Tabel 6.4 akan menunjukkan perhitungan manual cek deteksi *error* pada data suhu 40°C menggunakan metode *Hamming Code*.

**Tabel 6.4 Perhitungan manual deteksi *error* pada data suhu 40°C**

Data	P1	P2	D3	P4	D5	D6	D7	P8	D9	D10	Px
10110100000	1	0	1	1	0	1	0	0	0	0	0
P1	$P1 \oplus D3 \oplus D5 \oplus D7 \oplus D9 = 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 = 0$										
P2	$P2 \oplus D3 \oplus D6 \oplus D7 \oplus D10 = 0 \oplus 1 \oplus 1 \oplus 0 \oplus 0 = 0$										
P4	$P4 \oplus D5 \oplus D6 \oplus D7 = 1 \oplus 0 \oplus 1 \oplus 0 = 0$										
P8	$P8 \oplus D9 \oplus D10 = 0 \oplus 0 \oplus 0 = 0$										
Cek Error	$(1 \times P1) + (2 \times P2) + (4 \times P4) + (8 \times P8) = (1 \times 0) + (2 \times 0) + (4 \times 0) + (8 \times 0) = 0$										
Data encode	10110100000 = no error										
Data suhu	D3,D5,D6,D7,D9,10 = 101000 = 40										

**Tabel 6.5 Hasil pengujian non fungsional pada suhu 12°C**

No	Nama Pengujian	Data Berhasil Terkirim	Hasil decode	Screenshoot
1	Pengujian ke-1	Ya	No error	

2	Pengujian ke-2	Ya	No error	Data ke 2 01111000 : No error Data asli: 1100 = 12C
3	Pengujian ke-3	Ya	No error	Data ke 3 01111000 : No error Data asli: 1100 = 12C
4	Pengujian ke-4	Ya	No error	Data ke 4 01111000 : No error Data asli: 1100 = 12C
5	Pengujian ke-5	Ya	No error	Data ke 5 01111000 : No error Data asli: 1100 = 12C
6	Pengujian ke-6	Ya	No error	Data ke 6 01111000 : No error Data asli: 1100 = 12C
7	Pengujian ke-7	Ya	No error	Data ke 7 01111000 : No error Data asli: 1100 = 12C
8	Pengujian ke-8	Ya	No error	Data ke 8 01111000 : No error Data asli: 1100 = 12C
9	Pengujian ke-9	Ya	No error	Data ke 9 01111000 : No error Data asli: 1100 = 12C
10	Pengujian ke-10	Ya	No error	Data ke 10 01111000 : No error Data asli: 1100 = 12C

Pada data ketiga yaitu data suhu 12°C yang ditunjukkan pada Tabel 6.5 berhasil diterima dan ditampilkan dengan status tidak ada *error* yang terdeteksi. Data yang berjumlah 10 gambar pengujian ini dapat menampilkan data secara stabil sama seperti pada kedua data sebelumnya. Selanjutnya adalah Tabel 6.6 yang menunjukkan perhitungan manual cek deteksi *error* metode *Hamming Code* pada data 12°C.

**Tabel 6.6 Perhitungan manual deteksi *error* pada data suhu 12°C**

Data	P1	P2	D3	P4	D5	D6	D7	Px
01111000	0	1	1	1	1	0	0	0
P1	$P1 \oplus D3 \oplus D5 \oplus D7 = 0 \oplus 1 \oplus 1 \oplus 0 = 0$							
P2	$P2 \oplus D3 \oplus D6 \oplus D7 = 1 \oplus 1 \oplus 0 \oplus 0 = 0$							
P4	$P4 \oplus D5 \oplus D6 \oplus D7 = 1 \oplus 1 \oplus 0 \oplus 0 = 0$							
Cek <i>error</i>	$(1 \times P1) + (2 \times P2) + (4 \times P4) = (1 \times 0) + (2 \times 0) + (4 \times 0) = 0$							
Data <i>encode</i>	01111000 = no error							
Data suhu	D3,D6,D6,D7 = 1100 = 12							

Berdasarkan pengujian yang dilakukan, ketiga data tersebut dapat menampilkan data suhu yang dikirim. Antara perhitungan pada sistem dan perhitungan manual menggunakan metode *Hamming Code* dapat membuktikan bahwa algoritma ini dapat melakukan *encode* dan *decode* data yang tepat.

### 6.3 Pengujian Data Error

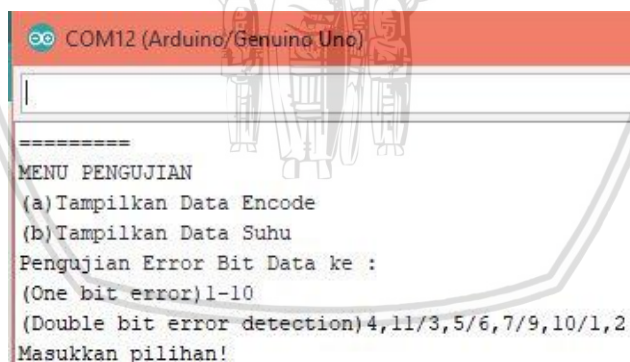
Pengujian data *error* dilakukan antar *Arduino Uno* dengan pin yang berbeda sesuai pada bagian perancangan. Pengujian dilakukan dengan mengubah data *encode* yang diterima.

#### 6.3.1 Tujuan Pengujian

Tujuan dari pengujian ini yaitu untuk mengetahui apakah sistem yang dirancang dapat melakukan deteksi dan koreksi *error* pada data *encode* yang diterima. Serta untuk mengetahui apakah setiap posisi data yang mengalami *error* dapat dideteksi atau dikoreksi. Pengujian pada penelitian ini akan memasukkan 5 posisi *error* pada data untuk 2 jenis *error* yang berbeda pada data suhu yang dikirim.

#### 6.3.2 Prosedur Pengujian

Berdasarkan perancangan dan implementasi yang dilakukan untuk pengujian data *error*, Tampilan menu pengujian pada serial monitor ditunjukkan pada Gambar 6.4.



**Gambar 6.4 Menu pengujian data *error* pada serial monitor**

Pada Gambar 6.4, terdapat beberapa pilihan pada menu pengujian yang akan menjadi data masukan untuk melakukan pengujian data *error*. Penjelasan tiap-tiap pilihan tersebut yaitu :

a. Tampilkan Data *Encode*.

Menu ini akan menampilkan data *encode* yang dikirim sebelum melakukan pengujian data *error* yang dimasukkan pada data *encode*.

b. Tampilkan Data Suhu.

Pilihan ini akan menampilkan data suhu yang dalam bentuk bilangan decimal.



c. *One bit error* (1-10).

Pada menu ini, akan memasukkan data yang berupa posisi *error* data. Pilihan ini dimasukkan dalam bentuk angka pada serial monitor. Kemudian satu posisi yang dimasukkan akan diganti datanya dengan format bilangan biner.

d. *Double bit error detection* (3,5/7,9,10,11/2,4/3,4).

Pilihan menu yang terakhir ini akan memasukkan data yang berupa posisi *error* data dengan 2 posisi.

Tahapan prosedur yang dilakukan untuk pada pengujian ini yaitu :

1. Menghubungkan sensor DHT11 pada mikrokontroler *Arduino Uno* yang berperan sebagai pengirim dan menghubungkan antar *Arduino Uno* pada pin yang telah dituliskan pada bab perancangan
2. Data suhu yang didapatkan diubah menjadi biner 7 bit.
3. Kemudian, data tersebut diencode dengan metode *Hamming Code* (7,12) dan dikirim.
4. Pada bagian penerima, memasukkan pilihan pengujian pada serial monitor *Arduino IDE* penerima yang ditampilkan.
5. Memilih pilihan a dan b untuk mengetahui data *encode* yang diterima.
6. Memasukkan pilihan *one bit error* atau *double bit error detection* untuk mengubah data pada posisi yang telah ditentukan pada pilihan.

### 6.3.3 Hasil dan Analisis Pengujian

Pengujian data *error* yang pertama akan menggunakan data suhu 24°C. Pada Gambar 6.5 dan Tabel 6.7 yang akan menunjukkan tentang data pengujian.

```
(b) Pengujian ke 5
= 24 C
(a) Pengujian ke 6
Data encode: 100001100001
Masukkan pilihan!
```

Gambar 6.5 Data pengujian pertama

Tabel 6.7 Pengujian satu bit *error* data pada data suhu 24°C

No	Nama Pengujian	Posisi Error	Screenshoot	Deteksi Error	Koreksi Error
1	Pengujian ke-1	2	Masukkan pilihan! (a) Pengujian ke 1 Data encode: 100001100001 Masukkan pilihan! 2 Data asli: 100001100001 Data pengujian: 110001100001 one bit error detected.. position: 2 Error Correction: 100001100001 0011000 = 24C	Ya	Ya

2	Pengujian ke-2	4	4 Data asli: 100001100001 Data pengujian: 100101100001 one bit error detected.. position: 4 Error Correction: 100001100001 0011000 = 24C	Ya	Ya
3	Pengujian ke-3	9	9 Data asli: 100001100001 Data pengujian: 100001101001 one bit error detected.. position: 9 Error Correction: 100001100001 0011000 = 24C Masukan pilihan!	Ya	Ya
4	Pengujian ke-4	1	Masukan pilihan! 1 Data asli: 100001100001 Data pengujian: 000001100001 one bit error detected... position: 1 Error Correction: 100001100001 0011000 = 24C Masukan pilihan!	Ya	Tidak
5	Pengujian ke-5	8	8 Data asli: 100001100001 Data pengujian: 100001110001 one bit error detected... position: 8 Error Correction: 100001100001 0011000 = 24C Masukan pilihan!	Ya	Tidak

Berdasarkan Tabel 6.7, satu bit *error* hasil perubahan masukan dapat dideteksi oleh metode *Hamming Code*. Tabel analisis perhitungan manual hasil implementasi akan ditunjukkan pada Tabel 6.8 :

**Tabel 6.8 Satu bit *error* posisi 4 pada data 24°C**

Posisi Data	P1	P2	D3	P4	D5	D6	D7	P8	D9	D10	D11	Px
Data yang dikirim	1	0	0	0	0	1	1	0	0	0	0	1
Data yang diterima	1	0	0	1	0	1	1	0	0	0	0	1
P1	$P1 \oplus D3 \oplus D5 \oplus D7 \oplus D9 \oplus D11 = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 \oplus 0 = 0$											
P2	$P2 \oplus D3 \oplus D6 \oplus D7 \oplus D10 \oplus D11 = 0 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 0 = 0$											
P4	$P4 \oplus D5 \oplus D6 \oplus D7 = 1 \oplus 0 \oplus 1 \oplus 1 = 1$											
P8	$P8 \oplus D9 \oplus D10 \oplus D11 = 0 \oplus 0 \oplus 0 \oplus 0 = 0$											
Cek <i>error</i>	$(1 \times P1) + (2 \times P2) + (4 \times P4) + (8 \times P8) = (1 \times 0) + (2 \times 0) + (4 \times 1) + (8 \times 0) = 4$											
Px	$P1 \oplus P2 \oplus D2 \oplus D3 \oplus D4 \oplus D5 \oplus D6 \oplus D7 \oplus P8 \oplus D9 \oplus D10 \oplus D11 \oplus Px = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 \oplus 0 \oplus 1 = 1$											
Data encode	100101100001 = <i>error 1 bit</i> pada posisi 4											

Hasil perhitungan manual pada Tabel 6.8 dan pengujian yang dilakukan menunjukkan kesamaan hasil yaitu dapat mendeteksi pada posisi data yang diubah nilainya pada data suhu 24°C. Selanjutnya adalah pengujian dengan deteksi *error* lebih dari satu posisi. Tabel pengujian dua *bit error* yang dilakukan sebanyak 5 kali pengujian pada data suhu 24°C ditunjukkan pada Tabel 6.9.

**Tabel 6.9 Pengujian dua bit *error* pada suhu 24°C**

No	Nama Pengujian	Posisi Error	Screenshoot	Deteksi Error	Koreksi Error
1	Pengujian ke-1	9 dan 10	posisi 9 dan 10 error data pengujian: 100001101101 double error detected.. Masukan pilihan!	Ya	Tidak

2	Pengujian ke-2	6 dan 7	Masukkan pilihan! posisi 6 dan 7 error data pengujian: 100000000001 double error detected..	Ya	Tidak
3	Pengujian ke-3	3 dan 5	posisi 3 dan 5 error data pengujian: 101011100001 double error detected.. Masukan pilihan!	Ya	Tidak
4	Pengujian ke-4	4 dan 11	(a) Pengujian ke 3 Data encode: 100001100001 Masukkan pilihan! posisi 4 dan 11 error data pengujian: 100101100011 double error detected.. .....	Ya	Tidak
5	Pengujian ke-5	1 dan 2	= 24 °C posisi 1 dan 2 error data pengujian: 010001100001 double error detected.. Masukan pilihan!	Ya	Tidak

Berdasarkan pengujian yang dilakukan pada Tabel 6.9, sistem dapat melakukan deteksi 2 bit error pada data encode yang mengalami perubahan. Tabel 6.10 adalah salah satu contoh perhitungan manual metode *Hamming Code* dalam mendeteksi 2 bit error.

**Tabel 6.10 Pengujian 2 bit error posisi 9 dan 10 pada data 24°C**

Posisi Data	P1	P2	D3	P4	D5	D6	D7	P8	D9	D10	D11	Px
Data yang dikirim	1	0	0	0	0	1	1	0	0	0	0	1
Data yang diterima	1	0	0	0	0	1	1	0	1	1	0	1
P1	$P1 \oplus D3 \oplus D5 \oplus D7 \oplus D9 \oplus D11 = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$											
P2	$P2 \oplus D3 \oplus D6 \oplus D7 \oplus D10 \oplus D11 = 0 \oplus 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 1$											
P4	$P4 \oplus D5 \oplus D6 \oplus D7 = 0 \oplus 0 \oplus 1 \oplus 1 = 0$											
P8	$P8 \oplus D9 \oplus D10 \oplus D11 = 0 \oplus 1 \oplus 1 \oplus 0 = 0$											

Cek error	$(1 \times P1) + (2 \times P2) + (4 \times P4) + (8 \times P8) = (1 \times 1) + (2 \times 1) + (4 \times 0) + (8 \times 0) = 3$
Px	$P1 \oplus P2 \oplus D2 \oplus D3 \oplus D4 \oplus D5 \oplus D6 \oplus D7 \oplus P8 \oplus D9 \oplus D10 \oplus D11 \oplus$ $Px = 1 \oplus 0 \oplus 0 \oplus 0 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 0$
Data encode	100001101101 = 2 bit error

Pada Tabel 6.10 yang merupakan perhitungan manual menunjukkan kesamaan hasil dengan proses pengujian yang dilakukan. Metode *Hamming Code* dapat mendeteksi 2 bit error pada data yang diubah.

Pengujian data kedua menggunakan data suhu 12°C. Pada Gambar 6.6 akan menunjukkan tampilan pengujian dengan data suhu tersebut.

```
Masukkan pilihan!
(a) Pengujian ke 1
Data encode: 010100111001
Masukkan pilihan!
b
(b) Pengujian ke 2
= 12 C
```

Gambar 6.6 Data pengujian kedua

Tabel 6.11 Pengujian error data pada data suhu 12°C

No	Nama Pengujian	Posisi Error	Screenshoot	Deteksi Error	Koreksi Error
1	Pengujian ke-1	10	10 Data asli: 010100111001 Data pengujian: 010100111101 one bit error detected.. position: 10 Error Correction: 010100111001 0001100 = 12C Masukan pilihan!	Ya	Ya
2	Pengujian ke-2	11	Masukan pilihan! 11 Data asli: 010100111001 Data pengujian: 01010011101 one bit error detected.. position: 11 Error Correction: 01010011100 0001100 = 12C	Ya	Ya



3	Pengujian ke-3	6	Masukan pilihan! 6 Data asli: 010100111001 Data pengujian: 010101111001 one bit error detected.. position: 6 Error Correction: 010100111001 0001100 = 12C	Ya	Ya
4	Pengujian ke-4	5	5 Data asli: 010100111001 Data pengujian: 010101111001 one bit error detected.. position: 5 Error Correction: 010100111001 0001100 = 12C Masukan pilihan!	Ya	Ya
5	Pengujian ke-5	3	Masukan pilihan! 3 Data asli: 010100111001 Data pengujian: 011100111001 one bit error detected.. position: 3 Error Correction: 010100111001 0001100 = 12C	Ya	Ya

Berdasarkan pengujian pada Tabel 6.11, data suhu bernilai 12°C yang diubah data *encode* nya pada 5 posisi tersebut dapat dideteksi posisinya dan dilakukan koreksi pada kelima posisi tersebut. Analisis perhitungan manual dari salah satu data yang mengalami 1 bit *error* akan ditunjukkan pada Tabel 6.12.

**Tabel 6.12 Perhitungan 1 bit error posisi 10 pada data 12°C**

Posisi Data	P1	P2	D3	P4	D5	D6	D7	P8	D9	D10	D11	Px
Data yang dikirim	0	1	0	1	0	0	1	1	1	0	0	1
Data yang diterima	0	1	0	1	0	0	1	1	1	1	0	1
P1	$P1 \oplus D3 \oplus D5 \oplus D7 \oplus D9 \oplus D11 = 0 \oplus 0 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 0$											
P2	$P2 \oplus D3 \oplus D6 \oplus D7 \oplus D10 \oplus D11 = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$											
P4	$P4 \oplus D5 \oplus D6 \oplus D7 = 1 \oplus 0 \oplus 0 \oplus 1 = 0$											
P8	$P8 \oplus D9 \oplus D10 \oplus D11 = 1 \oplus 1 \oplus 1 \oplus 0 = 1$											
Cek error	$(1 \times P1) + (2 \times P2) + (4 \times P4) + (8 \times P8) = (1 \times 0) + (2 \times 1) + (4 \times 0) + (8 \times 1) = 10$											

Px	$P1 \oplus P2 \oplus D2 \oplus D3 \oplus D4 \oplus D5 \oplus D6 \oplus D7 \oplus P8 \oplus D9 \oplus D10 \oplus D11 \oplus$ $Px = 0 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 1$
Data encode	010100111101 = <i>error 1 bit</i> pada posisi 10

Perhitungan manual pada Tabel 6.12 dan hasil pengujian pada Tabel 6.11 memiliki kesamaan hasil dalam melakukan deteksi *error* menggunakan metode *Hamming Code* yang diimplementasikan.

Selanjutnya, merupakan pengujian 2 bit *error* pada data suhu 12°C yang dilakukan sebanyak 5 kali pada Tabel 6.13.

**Tabel 6.13 Pengujian dua *bit error* pada suhu 12°C**

No	Nama Pengujian	Posisi Error	Screenshoot	Deteksi Error	Koreksi Error
1	Pengujian ke-1	4 dan 11	posisi 4 dan 11 error data pengujian: 010000111011 double error detected.. Masukan pilihan!	Ya	Tidak
2	Pengujian ke-2	9 dan 10	(a) Pengujian ke 3 Data encode: 010100111001 Masukkan pilihan! posisi 9 dan 10 error data pengujian: 010100110101 double error detected..	Ya	Tidak
3	Pengujian ke-3	1 dan 2	(a) Pengujian ke 4 Data encode: 010100111001 Masukkan pilihan! posisi 1 dan 2 error data pengujian: 100100111001	Ya	Tidak
4	Pengujian ke-4	3 dan 5	Masukan pilihan! posisi 3 dan 5 error data pengujian: 01110111001 double error detected.. Masukan pilihan!	Ya	Tidak

5	Pengujian ke-5	6 dan 7	Masukan pilihan! posisi 6 dan 7 error data pengujian: 01011111001 double error detected.. Masukan pilihan!	Ya	Tidak
---	----------------	---------	---	----	-------

Dua data yang diubah secara acak pada pengujian yang dilakukan pada Tabel 6.13 dapat dideteksi oleh metode *Hamming Code*. Hasil pengujian pada sistem memiliki kesamaan pada hasil perhitungan manual pada Tabel 6.14.

**Tabel 6.14 Dua *bit error* posisi 3 dan 5 pada data 12<sup>o</sup>c**

Posisi Data	P1	P2	D3	P4	D5	D6	D7	P8	D9	D10	D11	Px
Data yang dikirim	0	1	0	1	0	0	1	1	1	0	0	1
Data yang diterima	0	1	1	1	1	0	1	1	1	0	0	1
P1	$P1 \oplus D3 \oplus D5 \oplus D7 \oplus D9 \oplus D11 = 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 0$											
P2	$P2 \oplus D3 \oplus D6 \oplus D7 \oplus D10 \oplus D11 = 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 0 = 1$											
P4	$P4 \oplus D5 \oplus D6 \oplus D7 = 1 \oplus 1 \oplus 0 \oplus 1 = 1$											
P8	$P8 \oplus D9 \oplus D10 \oplus D11 = 1 \oplus 1 \oplus 0 \oplus 0 = 0$											
Cek error	$(1 \times P1) + (2 \times P2) + (4 \times P4) + (8 \times P8) = (1 \times 0) + (2 \times 1) + (4 \times 1) + (8 \times 0) = 6$											
Px	$P1 \oplus P2 \oplus D2 \oplus D3 \oplus D4 \oplus D5 \oplus D6 \oplus D7 \oplus P8 \oplus D9 \oplus D10 \oplus D11 \oplus Px = 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 \oplus 0 \oplus 0 \oplus 1 = 0$											
Data encode	011110111001 = error 2 bit											

Berdasarkan pengujian 2 data pengujian yang disisipi oleh 2 jenis *error* dengan 10 kali pengujian, metode *Hamming Code* dapat mendeteksi jenis *error* tersebut. Namun, dalam melakukan koreksi data, metode *Hamming Code* hanya mampu melakukan koreksi 1 bit pada deteksi 1 bit data *error*. *Parity bit* tambahan yaitu Px sangat berpengaruh dalam mendeteksi jenis *error*, apakah data *encode* mengalami *single error* atau *burst error*. Nilai *parity bit* tambahan ini tidak berpengaruh dalam operasi logika *XOR* pada *parity bit* P1, P2, P4, dan P8. Semakin

banyak jumlah data yang dikirim, maka akan semakin banyak juga *parity bit* yang ditambahkan sesuai dengan aturan metode *Hamming Code*.

## 6.4 Pengujian Waktu Metode *Hamming Code*

### 6.4.1 Tujuan Pengujian

Pengujian terakhir dilakukan untuk mengetahui waktu yang dibutuhkan untuk mengeksekusi metode *Hamming Code* dalam proses *encode* dan *decode* data suhu. Pengujian dilakukan dengan menggunakan 2 data yang memiliki jumlah data *encode* yang berbeda.

### 6.4.2 Prosedur Pengujian

1. Menghubungkan sensor DHT11 pada mikrokontroler *Arduino Uno* yang berperan sebagai pengirim.
2. Menghubungkan mikrokontroler *Arduino Uno* dengan laptop menggunakan kabel *usb*.
3. Menambahkan fungsi *millis()* pada bagian pengirim untuk mengetahui waktu yang dibutuhkan untuk *encode* data dan pada bagian penerima untuk mengetahui waktu yang dibutuhkan untuk *decode* data dengan metode *Hamming Code*.
4. Meng-*upload* kode program pada bagian pengirim dan dilakukan analisis waktu yang ditampilkan pada serial monitor.
5. Setelah selesai melakukan analisis pada langkah nomor 4, menghubungkan Pin *UART* pada bagian pengirim dengan pin digital pada penerima menggunakan kabel *jumper*.
6. Selanjutnya meng-*upload* kode program untuk mengirim data *encode* dan *decode*.
7. Melakukan analisis waktu *decode* yang ditampilkan pada serial monitor pada bagian penerima.

### 6.4.3 Hasil dan Analisis Pengujian

Hasil pengujian proses *encode* data menggunakan metode *Hamming Code* pada data suhu yang terbaca akan ditunjukkan pada 2 subbab selanjutnya.

6.4.3.1 Pengujian Waktu *Encode* DataTabel 6.15 Data pengujian *encode* data suhu 28°C

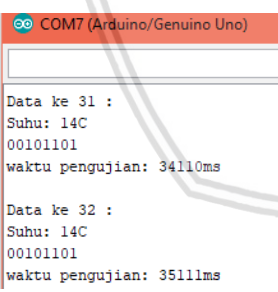
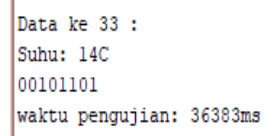
No	Screenshoot Pengujian	Selisih Waktu	Selisih delay (1000ms)
1	Data ke 1 : Suhu: 0C 0000000 waktu pengujian: 0ms	0 ms	0s
2	Data ke 2 : Suhu: 0C 0000000 waktu pengujian: 1000ms	$1000 - 0 = 1000\text{ms}$	0ms
3	Data ke 3 : Suhu: 28C 0010110001 waktu pengujian: 2273ms	$2273 - 1000 = 1273\text{ms}$	273ms
4	Data ke 4 : Suhu: 28C 0010110001 waktu pengujian: 3274ms	$3274 - 2273 = 1001\text{ms}$	1ms
5	Data ke 5 : Suhu: 28C 0010110001 waktu pengujian: 4546ms	$4564 - 3274 = 1290\text{ms}$	190ms
6	Data ke 6 : Suhu: 28C 0010110001 waktu pengujian: 5548ms	$5548 - 4564 = 984\text{ms}$	16ms
7	Data ke 7 : Suhu: 28C 0010110001 waktu pengujian: 6820ms	$5548 - 6820 = 1272\text{ms}$	272ms



8	Data ke 8 : Suhu: 28C 0010110001 waktu pengujian: 7822ms	7822 – 6820 = 1002ms	2ms
9	Data ke 9 : Suhu: 28C 0010110001 waktu pengujian: 9094ms	7822 – 9094 = 1272ms	272ms
10	Data ke 10 : Suhu: 28C 0010110001 waktu pengujian: 10095ms	10095 – 9094 = 1001ms	1ms
$\Sigma$ Waktu selisih waktu <i>encode</i> Jumlah data		$\frac{1027}{10} = 102,7\text{ms}$	

Pada Tabel 6.15, data suhu 28°C dilakukan proses *encode* menjadi 10 bit data dengan waktu pemrosesan yang paling cepat yaitu 1 ms atau 0,001 detik. Sedangkan Waktu pemrosesan yang paling lama 273 ms atau 0,273 detik yang berjumlah 2 data. Sedangkan pengujian selanjutnya dilakukan pada data yang berbeda yaitu pada data suhu dibawah 16°C yang mempunyai jumlah bit sebanyak 4 bit jika dibinerkan. Tabel 6.16 akan menunjukkan pengujian tersebut.

**Tabel 6.16 Data pengujian *encode* data suhu 14°C – 13°C**

No	Screenshoot Pengujian	Selisih Waktu	Selisih delay (1000ms)
1		35111 – 34110 = 1001 ms	1 ms
2		36383 – 35111 = 1272ms	272ms

3	Data ke 34 : Suhu: 14C 00101101 waktu pengujian: 37385ms	$37385 - 36383 = 1002\text{ms}$	2ms
4	Data ke 35 : Suhu: 13C 10101010 waktu pengujian: 38658ms	$38658 - 37385 = 1273\text{ms}$	273ms
5	Data ke 36 : Suhu: 13C 10101010 waktu pengujian: 39659ms	$39659 - 38658 = 1001\text{ms}$	1ms
6	Data ke 37 : Suhu: 13C 10101010 waktu pengujian: 40931ms	$40931 - 39659 = 1272\text{ms}$	272ms
7	Data ke 38 : Suhu: 13C 10101010 waktu pengujian: 41932ms	$41932 - 40931 = 1000\text{ms}$	0 ms
8	Data ke 39 : Suhu: 13C 10101010 waktu pengujian: 43205ms	$43205 - 41932 = 1272\text{ms}$	272ms
9	Data ke 40 : Suhu: 13C 10101010 waktu pengujian: 44207ms	$44207 - 43205 = 1002\text{ms}$	2ms
10	Data ke 41 : Suhu: 13C 10101010 waktu pengujian: 45207ms <input type="checkbox"/> Autoscroll	$45207 - 44207 = 1000\text{ms}$	0ms

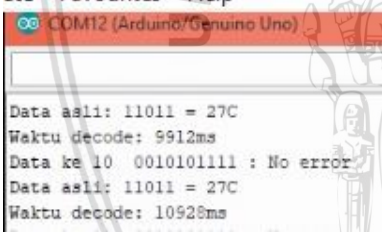
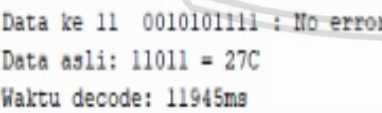
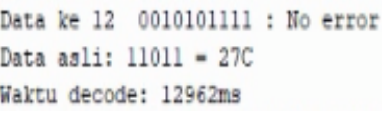
$\frac{\Sigma \text{Waktu selisih waktu } encode}{\text{Jumlah data}}$	$\frac{1095}{10} = 109,5\text{ms}$
--	------------------------------------

Pada Tabel 6.16, pengujian yang dilakukan mendapatkan jumlah data suhu yang berbeda-beda antara 14°C – 13°C. Perbedaan data yang dihasilkan dari pembacaan sensor suhu DHT11 dapat mempengaruhi waktu yang dibutuhkan untuk dilakukan proses *encode* data. Sehingga terdapat 4 data dengan jumlah waktu *encode* paling lama 273 ms atau 0,273 detik dan terdapat 2 data dengan jumlah *encode* paling singkat yaitu 0ms.

Banyaknya jumlah data yang dilakukan proses *encode* dengan menggunakan metode *Hamming Code* juga mempengaruhi lama waktu yang dibutuhkan. Pada Tabel 6.15, terdapat 4 data yang membutuhkan waktu kurang dari 100 ms dalam melakukan proses *encode* dengan metode *Hamming Code*, sedangkan pada Tabel 6.16, terdapat 6 data yang membutuhkan waktu kurang dari 100 ms dalam melakukan proses *encode* data.

#### 6.4.3.2 Pengujian Waktu *Decode* Data

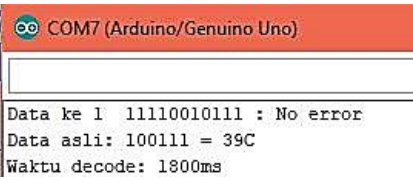
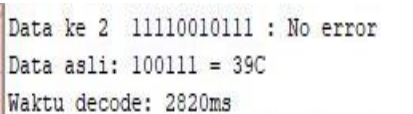
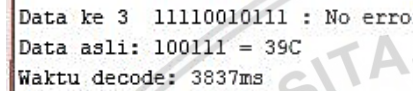
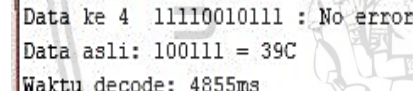
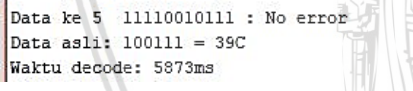
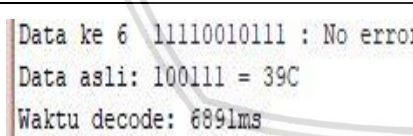
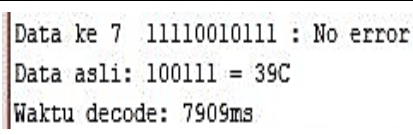
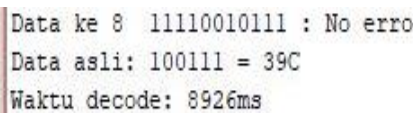
**Tabel 6.17 Data pengujian *decode* data suhu 27°C - 26°C**

No	Screenshoot Pengujian	Selisih Waktu	Selisih delay (1000ms)
1		10928 - 9912 = 1016 ms	16ms
2		11954 - 10928 = 1026ms	26ms
3		12962 - 11954 = 1008ms	8ms

4	Data ke 13 0010101111 : No error Data asli: 11011 = 27C Waktu decode: 13980ms	13980 – 12962 = 1018ms	18ms
5	Data ke 14 0010101111 : No error Data asli: 11011 = 27C Waktu decode: 14998ms	14998 – 13980 = 1018ms	18ms
6	Data ke 15 0010101111 : No error Data asli: 11011 = 27C Waktu decode: 16016ms	16016 – 14998 = 1018ms	18ms
7	Data ke 17 1010101000 : No error Data asli: 11010 = 26C Waktu decode: 18034ms Data ke 18 1010101000 : No error Data asli: 11010 = 26C Waktu decode: 19052ms	19052 – 18034 = 1018ms	18ms
8	Data ke 19 1010101000 : No error Data asli: 11010 = 26C Waktu decode: 20069ms	20069 – 19052 = 1017ms	17ms
9	Data ke 20 1010101000 : No error Data asli: 11010 = 26C Waktu decode: 21087ms	21087 – 20069 = 1018ms	18ms
10	Data ke 21 1010101000 : No error Data asli: 11010 = 26C Waktu decode: 22105ms	22105 – 21087 = 1018ms	18ms
$\Sigma$ Waktu selisih waktu <i>encode</i> Jumlah data		$\frac{175}{10} = 17,5\text{ms}$	

Pada pengujian selanjutnya yang ditunjukkan pada Tabel 6.17, data suhu yang didapatkan yaitu antara 27°C – 26°C dilakukan proses *decode* data dari 10 bit data *encode* menjadi 5 bit data asli. Waktu pemrosesan yang dibutuhkan paling cepat sebanyak 8 ms atau 0,008 detik dan waktu yang dibutuhkan paling lama sebanyak 26 ms atau 0,0026 detik.

Tabel 6.18 Pengujian *decode* data suhu 39°C

No	Screenshoot Pengujian	Selisih Waktu	Selisih delay (1000ms)
1		1800 ms	800ms
2		$2820 - 1800 = 1020\text{ms}$	20ms
3		$3873 - 2820 = 1053\text{ms}$	53ms
4		$4855 - 3837 = 1018\text{ms}$	18ms
5		$5873 - 4855 = 1022\text{ms}$	22ms
6		$6891 - 5873 = 1018\text{ms}$	18ms
7		$7909 - 6891 = 1018\text{ms}$	18ms
8		$8926 - 7909 = 1017\text{ms}$	17ms



9	Data ke 9 11110010111 : No error Data asli: 100111 = 39C Waktu decode: 9944ms	9944 – 8926 = 1018ms	18ms
10	Data ke 10 11110010111 : No error Data asli: 100111 = 39C Waktu decode: 10961ms	10961 – 9944 = 1017ms	17ms
$\Sigma$ Waktu selisih waktu <i>encode</i> Jumlah data		$\frac{1001}{10} = 100,1\text{ms}$	

Pada pengujian terakhir yang ditunjukkan pada Tabel 6.18 data tersebut dilakukan proses *encode* sebanyak 11 *bit* data *encode*. Waktu paling cepat yang dibutuhkan dalam proses *encode* sebanyak 17 ms atau 0,017 detik. Sedangkan waktu paling lama yang dibutuhkan yaitu sebanyak 800ms atau 0,8 detik.

Berdasarkan percobaan waktu *decode* data yang dilakukan pada Tabel 6.17 dan Tabel 6.18, banyaknya jumlah bit yang ditambahkan *parity bit* pada proses *encode* data mempengaruhi waktu deteksi *error* yang dilakukan dengan menggunakan metode *Hamming Code*. Pada Tabel 6.17 jumlah data yang dilakukan proses *decode* dengan waktu kurang dari 20ms sebanyak 9 data, sedangkan pada Tabel 6.18 berjumlah 5 data. Sehingga semakin banyak jumlah *parity bit* yang ditambahkan maka akan bertambah pula waktu yang dibutuhkan untuk melakukan deteksi *error*.

## BAB 7 PENUTUP

Pada bab ini membahas tentang kesimpulan dan saran yang didapatkan dari hasil penelitian. Kesimpulan menjelaskan tentang hasil perancangan sampai dengan pengujian dan analisis, sedangkan saran menjelaskan tentang pengembangan sistem dalam penelitian ini kedepannya.

### 7.1 Kesimpulan

Kesimpulan didapatkan dengan memperhatikan rumusan masalah yang tercantum pada penelitian. Serta dengan mendapatkan hasil perancangan, implementasi, dan pengujian yang dilakukan. Kesimpulan yang didapatkan yaitu :

1. Pada penelitian ini, metode *Hamming Code* diimplementasikan pada *Arduino Uno* dengan model simulasi antar dua *Arduino Uno* dengan menggunakan data suhu sebagai objek penelitian. Berdasarkan pengujian fungsional dan nonfungsional yang dilakukan, penerapan metode *Hamming Code* dalam melakukan *encode* dan *decode* data yaitu dengan menggunakan operasi logika *XOR* pada tiap data dalam bentuk biner. Sehingga data suhu yang didapatkan dari sensor DHT11 harus terlebih dahulu diubah ke dalam bentuk biner. Kemudian, jumlah penambahan *parity bit* yang didapatkan dari operasi logika *XOR* ini tergantung pada jumlah data yang didapatkan. Penelitian ini, menggunakan 5 *parity bit* tambahan untuk jumlah maksimal *bit* yang ditambahkan.
2. Data suhu yang didapatkan dari sensor DHT11 berhasil dilakukan *encode* dan *decode* menggunakan metode *Hamming Code*. Pada pengujian nonfungsional yang dilakukan, hasil *encode* dan *decode* data sama dengan hasil perhitungan manual dengan metode *Hamming Code*. Selain itu pada pengujian data *error* dimana pengujian ini dilakukan dengan mengubah data yang diterima, deteksi *error* dengan metode *Hamming Code* dapat melakukan deteksi pada proses *decode* data di semua posisi. Serta dapat melakukan koreksi *error* jika terdeteksi 1 *bit error* data, dan dapat mendeteksi 2 *bit error* secara *burst error* pada data yang diterima. Banyaknya *parity bit* yang ditambahkan juga mempengaruhi waktu eksekusi yang dilakukan. Semakin banyak jumlah *parity bit* yang ditambahkan maka akan bertambah pula waktu yang dibutuhkan.
3. Rata-rata *delay* waktu *encode* data yang dihasilkan berdasarkan pengujian yang dilakukan pada 2 jenis data yang berbeda yaitu 102,7ms pada data 4 bit dan 109,5ms pada data 5 bit. Perubahan data suhu di dapat pada lingkungan yang berbeda juga dapat mempengaruhi proses *encode* data. Sedangkan rata-rata *delay* waktu *decode* data yang dihasilkan pada 2 jenis data yang berbeda yaitu 17,5ms pada data 10 bit dan 100,1ms pada data 11 bit. Banyaknya jumlah data yang dilakukan proses *decode* juga dapat mempengaruhi lamanya waktu yang dibutuhkan.

## 7.2 Saran

Berdasarkan kesimpulan yang telah didapatkan, beberapa saran yang dihasilkan dari penulis untuk mengembangkan penelitian ini kedepannya :

1. Metode *Hamming Code* dapat diimplementasikan pada *real* sistem seperti contohnya pada sistem IOT.
2. Metode *Hamming Code* dapat diimplementasikan dengan metode *Information Redudancy* lainnya pada sistem yang sama seperti metode *LFSR*, *cyclic codes*, dll untuk lebih dianalisis tentang kelebihan dan kelemahannya.



## DAFTAR PUSTAKA

- Arduino, 2017. *ARDUINO UNO REV3*. [Online]  
Available at: <https://store.arduino.cc/usa/arduino-uno-rev3>  
[Diakses 05 January 2018].
- BYU, 2003. Universal Asynchronous Receiver/Transmitter. pp. 2-20.
- Hamming, R., 1950. *The Bell System Technical Journal*. s.l.:American Telephone and Telegraph Company.
- Kharagpur, 2008. *Data Link Control*. 2nd penyunt. s.l.:s.n.
- Liu, T., 2017. *Digital Digital Digital Digital relative relative relative relative humidity humidity humidity humidity & temperature emperature emperature emperature sensor AM2302/DHT22*. [Online]  
Available at: <https://cdn-shop.adafruit.com/datasheets/Digital+humidity+and+temperature+sensor+AM2302.pdf>  
[Diakses 05 January 2018].
- Lubis, A. A. A., Poltak, S. & Arman, S., 2012. Perancangan Error Detection System And Error. *Jurnal USU*, I(1), pp. 1-2.
- Mahendra, G. R., Sari, W. M. & Meilani, T. N., 2016. SIMULASI DETEKSI BIT ERROR MENGGUNAKAN METODE HAMMING CODE. *Jurnal Dinamika Informatika*, V(2), pp. 1-3.
- Muhajir, F., Efendi, S. & Sutarman, 2016. DETEKSI DAN KOREKSI MULTI BIT ERROR DENGAN. *Jurnal Teknovasi*, III(2), pp. 2-3.
- Olofsson, M., 2005. *Error Control Coding*. s.l.:s.n.
- Setiawan, F. E. & Suryawan, F., 2014. SIMULASI KODE HAMMING, KODE BCH, DAN KODE REED SOLOMON UNTUK OPTIMALISASI FORWARD ERROR CORRECTION. *UMS ETD-db*, Issue 3, pp. 2-4.
- Singh, A. K., 2016. Error Detection and Correction by Hamming Code. *Error Detection and Correction by Hamming Code*, 1(1), p. 35.
- Singh, A. K., 2016. Error Detection and Correction by Hamming Code. *Error Detection and Correction by Hamming Code*, 1(35), p. 35.